Зборник радова Факултета техничких наука, Нови Сад

УДК: 4.43
ДОИ: https://doi.org/10.24867/35BE16Bulaja

Имплементација Elasticsearch клијента у ZIO екосистему

Implementation of Elasticsearch client in ZIO ecosystem

Димитрије Булаја, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *Овај рад описује израду и имплементацију ZIO Elasticsearch библиотеке, односно типски безбедног Elasticsearch клијента, природно интегрисаног у ZIO екосистем, употребом програмског језика Scala, ZIO окружења и других ZIO изворних библиотека.*

Кључне речи (три до пет): *ZIO, Elasticsearch, Scala, функционално програмирање*

Abstract – *This paper describes the development and implementation of the ZIO Elasticsearch library, a type-safe Elasticsearch client naturally integrated into the ZIO ecosystem, using the Scala programming language, the ZIO environment, and other ZIO-native libraries.*

Keywords: (three to five): *ZIO, Elasticsearch, Scala, functional programming*

НАПОМЕНА: Овај рад проистекао је из мастер рада чији ментор је био др Драган Ивановић, ред. проф.

1.УВОД

У свету дистрибуираних система и микросервисних архитектура, неопходна је технологија која омогућава брзу претрагу, анализу и обраду великих количина података, уз висок ниво доступности, капацитета, скалабилности и поузданости. Један од најзаступљенијих софтвера који испуњава ове захтеве јесте *Elasticsearch* [1]. *Elasticsearch* је моћан претраживачки и аналитички алат, изграђен на *Lucene* библиотеци [2], који омогућава ефикасну обраду и анализу различитих врста података.

Паралелно, *ZIO* екосистем [6], развијен у програмском језику *Scala,* омогућава типски безбедан и једноставан рад са ефектима, асинхроним операцијама, конкурентношћу и са руковањем грешкама, кроз јасан и функционалан стил програмирања.

До скоро није постојало адекватно нативно, идиоматско *ZIO* решење за интеграцију са *Elasticsearch*-ом. Због тога су програмери били приморани да развијају прилагођене интеграције, што је често водило ка комплекснијим системима, већем простору за грешке и смањеној поузданости.

Полазећи од те празнине, развијена је библиотека *ZIO Elasticsearch*, клијент који спаја *Elasticsearch* сервер са *ZIO* ефектним моделом. Библиотека обезбеђује типску сигурност приликом моделовања упита, агрегација, као и приликом стримовања података. Дизајнирана је да омогући једноставну употребу, високе перформансе, скалабилност и робусно управљање грешкама. Такође, обезбеђена је нативна интеграција са *ZIO* екосистемом.

Рад је организован у неколико поглавља. Након уводне мотивације, пружа се увид у теоријске основе и коришћене технологије. Затим је приказана имплементација библиотеке и њена употреба кроз малу демонстративну апликацију. На крају, у оквиру закључка, сумирани су резултати и могући правци будућег развоја.

2.ТЕОРИЈСКЕ ОСНОВЕ И КОРИШЋЕНЕ ТЕХНОЛОГИЈЕ

У овом поглављу биће описани теоријски концепти, дефиниције и технологије које представљају основу за имплементацију библиотеке. Разумевање ових појмова је неопходно како би се јасно схватила логика, архитектура и решења која су примењена током развоја *ZIO Elasticsearch* библиотеке.

2.1.Функционално програмирање

Функционално програмирање представља једну од програмских парадигми у развоју софтвера, код које је главна идеја употреба функција као основне јединице кода. Поставља се акценат на чисте функције, имутабилност и декларативни стил програмирања, тачније представља својеврсну рестрикцију на то како програм треба нешто да уради, а не на то шта тај програм треба да уради. Овај приступ олакшава тестабилност, предвидљивост и паралелизацију, што је од суштинског значаја у дистрибуираним системима, док конструкције као што су функције вишег реда и богати типски системи повећавају изражајност и безбедност кода [3][5].

2.2.Scala и JVM

Програмски језик *Scala* осмишљен је као обједињење објектно-оријентисаног и функционалног програмирања, које пружа програмерима могућност да користе оба стила у складу са потребама апликације. Захваљујући чињеници да се *Scala* извршава на *Java* виртуелној машини, омогућена је интеграција са постојећим *Java* екосистемом. Ова компатибилност омогућава приступ великом броју постојећих

библиотека и алата, што је допринело широј примени *Scala*-е у развоју модерних система.
Scala нуди концизну синтаксу и снажан типски систем, што омогућава писање изражајнијег и безбеднијег кода. Такође, *Scala* пружа могућност развоја апликација које захтевају паралелизам и високе перформансе, а омогућава и креирање апстракција, које смањују дуплирање кода и поједностављују одржавање софтвера [4].

2.3.*ZIO* екосистем

Основни концепт *ZIO* библиотеке јесте ”ефекат” (тип *ZIO Effect*), који енкапсулира програмске операције попут читања, писања или управљања грешкама. Сем тога, обезбеђује снажан модел за управљање ресурсима и омогућава програмирање без споредних ефеката, што додатно олакшава развој апликација [3].
ZIO тип репрезентује се као *ZIO[R, E, A]*, где су *R*, *E* и *A* редом: окружење, грешка и успешан исход. Оваква структура омогућава потпуно раздвојено управљање успешним и неуспешним исходима, чиме се знатно поједностављује обрада грешака [6].
Још један од кључних аспеката *ZIO* библиотеке јесте комбиновање и трансформација ефеката (путем метода *map*, *flatMap* и *zip*). Поред тога, *ZIO* обезбеђује и једноставно управљање паралелизмом, што значи да se више ефеката могу истовремено извршавати без потребе за ручним управљањем конкурентношћу [7].
Иако је првобитно био усмерен само на управљање ефектима, *ZIO* временом прераста оквире библиотеке и еволуира у комплетан екосистем. Тај екосистем укључује разне библиотеке које омогућавају лакшу интеграцију са екстерним системима и базама података, рад са разним врстама података и изградњу дистрибуираних апликација [6].

2.4.Elasticsearch и *Lucene*

Elasticsearch је дистрибуирани софтвер отвореног кода који омогућава претрагу и анализу података. Развијен је у *Java* програмском језику и заснива се на *Lucene* библиотеци. Омогућава брз приступ информацијама у реалном времену кроз *REST API* и *JSON* формат.
Захваљујући свом дистрибуираном складишту са шардовима и репликама, овај софтвер програмерима пружа могућност рада са разноврсним типовима података. Кластер се може хоризонтално скалирати, а сви чворови унутар кластера су потпуно независни и самостални. Оваквим дизајном елиминише се ”*single point of failure*” проблем и задржавају се високе перформансе.
Један од основних коцепата *Elasticsearch*-а је структура података ”инвертовани индекс”, коју *Elasticsearch*, помоћу *Lucene* библиотеке, користи за ефикасну и брзу претрагу текста [8].
Lucene је библиотека за индексирање и претрагу текста, коју је развио *Apache Software Foundation* [9]. Ради побољшања перформанси претраживања, *Lucene* библиотека пре него што крене са индексирањем, врши претпроцесирање података, а потом те претпроцесиране податке складишти у инвертованом индексу. Ова библиотека такође подржава и различите типове претрага, од једноставних до веома комплексних упита и филтера.

2.5.Fluent *API*

Fluent API је дизајн шаблон који омогућава ланчано позивање метода над истим објектом, где свако ланчање враћа нову инстанцу и тиме одржава читљив, типски безбедан код, без сувишних променљивих. Често се користи у *Query Builder*-има, конфигурационим фајловима и спецификацијама пословне логике, где је потребна јасна семантика при конструисању сложених објеката.

2.6.Streaming у *Elasticsearch*-у

Streaming омогућава да се велики скупови резултата обрађују постепено, уместо да се цео сет учита одједном у меморију, што је кључно у дистрибуираним системима који раде у реалном времену. *Elasticsearch* нуди два механизма за *streaming*. Први механизам је *scroll*, он отвара серверски контекст и преко *scrollId* идентификатора враћа резултате део по део, све док се контекст експлицитно не затвори [10]. Други механизам се назива *search_after,* користи *stateless* приступ, који захтева сортиране резултате, док у свакој враћеној страници резултата шаље и вредности потребне за наставак претраге. У пракси се *search_after* често комбинује са *Point in Time* (*PIT*), који ”замрзава” индекс у тренутном стању. *PIT* гарантује да ће све странице резултата одражавати исту верзију података, чак и у случају да се у међувремену упишу нови документи, или бришу постојећи [11].

3.ИМПЛЕМЕНТАЦИЈА

Библиотека *ZIO Elasticsearc*h омогућава идиоматски *ZIO* приступ *Elasticsearch*-у, пружајући реактивни модел рада са асинхроним операцијама, као и сигурно руковање ресурсима и грешкама. Библиотека подржава верзије 7 и 8 *Elasticsearch*-а, као и развој апликација у *Scala* програмском језику верзија 2 и 3.
Изворни код библиотеке је организован у складу са стандардном *SBT* структуром пројекта, где се изворни фајлови налазе унутар путање *src/main*, а тестови унутар путање *src/test*.
Архитектура библиотеке заснива се на принципима функционалног програмирања у *ZIO* екосистему са акцентом на тестабилност, декларативност и *Separation of Concerns* [12]. У основи библиотеке се налази апстракција *ElasticRequest*, која моделује сваки захтев ка *Elasticsearch*-у и апстракција *Executor*, која извршава захтеве и одговорна је за комуникацију са *Elasticsearch* сервером. Кроз *ZIO* слојеве (*ZLayer* [14]) омогућена је једноставна примена *Dependency injection*-а [13] и изузетно флексибилно конфигурисање окружења.
Сви захтеви се моделују као класе, које наслеђују апстрактни тип *ElasticRequest[A],* где *A* представља енкапсулирани тип који се враћа приликом извршавања захтева. Корисници захтеве не креирају директно, већ преко јавног *API*-ја, а након креирања, могу над њима позивати и одговарајуће методе које постављају опционе параметре.
Executor апстракција дефинише како се захтеви извршавају и како се обрађују резултати, а *HTTPExecutor* представља конкретну имплементацију,

која преко *STTP* клијента комуницира са *Elasticsearch* сервером. Апстракција *Elasticsearch* (листинг 1) служи као главна улазна тачка за кориснике, а све функције унутар те апстракције су дефинисане помоћу *ZIO* ефеката.

```
trait Elasticsearch {
  def execute[A](request: ElasticRequest[A]): Task[A]

  def stream(request: SearchRequest): Stream[Throwable,
Item]

  def stream(request: SearchRequest, config: StreamConfig):
Stream[Throwable, Item]

  def streamAs[A: Schema](request: SearchRequest):
Stream[Throwable, A]

  def streamAs[A: Schema](request: SearchRequest, config:
StreamConfig): Stream[Throwable, A]
}
```

Листинг 1. *Elasticsearch* апстракција

Један од кључних концепата у библиотеци јесте моделовање упита и агрегација према *Elasticsearch* серверу. *ElasticQuery* објекат пружа јавни *API*, који садржи функције за креирање различитих врста упита, док *ElasticAggregation* објекат садржи јавни *API* са функцијама за креирање различитих врста агрегација. Због тога што се упити и агрегације дефинишу као класе унутар *Scala* кода, обезбеђује се типска сигурност, боља читљивост и једноставна композиција сложенијих упита и агрегација из једноставнијих. Свака инстанца упита или агрегације може се обогатити додатним, опционим параметрима ланчаним позивањем метода над том инстацом.
Уместо да се ослања на ”ниске типове”, за представљање имена индекса или вредности рутирања, библиотека *ZIO Elasticsearch* користи специјализоване типове. Поред тога што омогућавају бољу контролу, типску сигурност и већу безбедност, специјализовани типови уводе и додатне валидације за сваки тип које се извршавају већ приликом компилације кода. На листингу 2 налази се дефиниција типа *IndexName*, који представља име индекса у *Elasticsearch*-у, дефинисан је над типом *String*, уз придружене валидације које одговарају правилима *Elasticsearch*-а.

```
object IndexName extends NewtypeCustom[String] {
  protected def validate(name: String) =
    IndexNameValidator.validate(name)
}
type IndexName = IndexName.Type
```

Листинг 2. Дефиниција *IndexName* типа за верзију 3 програмског језика *Scala*

Поред класичног рада са индексима и документима, библиотека омогућава и извршавање више операција над документима у оквиру истог захтева, помоћу *bulk API*-ја *Elasticsearch*-а. Овај механизам значајно убрзава рад, јер се више операција групише и шаље истовремено. У *ZIO Elasticsearch* библиотеци, за коришћење овог механизма предвиђена је функција *bulk* (листниг 3), која прихвата произвољан број захтева типа *BukableRequest* (апстракција коју наслеђују само захтеви који смеју бити укључени у *bulk* операцију).

```
final def bulk(requests: BulkableRequest[_]*): BulkRequest =
  Bulk(requests = Chunk.fromIterable(requests), index =
None, refresh = None, routing = None)
```

Листинг 3. Функција за креирање *Bulk* захтева

Постоје три врсте захтева чијим извшавањем корисници могу да врше претрагу и агрегирање: *SearchRequest*, *AggregateRequest* и *SearchAndAggregateRequest*. Сваки од захтева омогућава пагинацију, сортирање по пољима или скриптованој логици, филтрирање, додавање нових агрегација и *highlighting* за истицање делова текста.
Библиотека *ZIO Elasticsearch* подржава стриминг докумената кроз *ZStream* [15], користећи један од два доступна механизма у *Elasticsearch*-у: “*search_after*” и “*scroll*”. Оба приступа омогућавају постепено преузимање резултата претраге у виду стрима, чиме је омогућено обрађивање података без претходног учитавања целог скупа у меморију.
Сви захтеви ка *Elasticsearch* серверу у оквиру библиотеке *ZIO Elasticsearch* извршавају се преко *HttpExecutor*-а. Он омогућава слање сваког од постојећих захтева и обраду одговора са сервера у типизованом облику. Свака функција из *HttpExecutor*-а формира *HTTP* захтев, поставља заглавља, шаље сам захтев (преко *HTTP Client*-а) и декодира одговор у очекивани тип помоћу *JSON* декодера. У случају грешака, *HTTP* одговори се мапирају у доменске изузетке, омогућавајући јасну обраду грешака.

4.ДЕМОНСТРАЦИЈА КОРИШЋЕЊА БИБЛИОТЕКЕ

Да би се приказало коришћење библиотеке *ZIO Elasticsearch* у пракси, развијена је једноставна *backend* апликација која симулира рад са *GitHub* репозиторијумима. Апликација омогућава чување информација о репозиторијумима (назив, организација, број звездица…), претрагу по различитим критеријумима, као и креирање, измену и брисање докумената у *Elasticsearch* индексу. Користи се *REST* интерфејс написан у *Scala* програмском језику, док се библиотека *ZIO Elasticsearch* користи за комуникацију са *Elasticsearch* кластером.
За покретање апликације, довољно је прво покренути локални *Elasticsearch* сервер и након тога позвати команду: *./sbt “~example/reStart”*, која ће заправо и подићи саму апликацију.

```
def createAll(repositories: Chunk[GitHubRepo]): Task[Unit]
=
  for {
    routing <- routingOf(organization)
      <- elasticsearch.execute(
         ElasticRequest
           .bulk(repositories.map { repository =>
             ElasticRequest.create[GitHubRepo](Index,
DocumentId(repository.id), repository)
           }: _*).routing(routing)
       )
  } yield ()
```

Листинг 4. Попуњавање *Elasticsearch* индекса

Приликом покретања апликације, прво се, уколико постоји, аутоматски обрише постојећи *Elasticsearch* индекс, након чега се индекс изнова креира и попуњава подацима коришћењем *bulk* операције са угњежденим *create* операцијама (листинг 4). Основни слој за рад са базом представља класа *RepositoriesElasticsearch*. Она користи *ZIO Elasticsearch* библиотеку за креирање и извршавање различитих типова захтева. Овај слој, уз помоћ библиотеке, омогућава креирање појединачних докумената, креирање више докумената одједном, ажурирање докумената, брисање по идентификатору, добављање свих докумената, добављање појединачних докумената и претрагу са пагинацијом.
Претрага је реализована кроз функцију *search* (листинг 4), којој се сем жељеног упита, прослеђују и параметри *from* і *size*, који се користе за пагинацију. Функција извршава захтев *Search* добијен *search* функцијом из библиотеке, односно из *ElasticRequest* класе. Као и код добављања докумената, над повратном вредности извршавања захтева, позива се метода ”*.documentAs[GitHubRepo]*” из библиотеке, која омогућава да документи који се добављају буду очекиваног типа, а не генерички.

```
def search(query: ElasticQuery[_], from: Int, size: Int):
Task[Chunk[GitHubRepo]] =
  elasticsearch.execute(ElasticRequest.search(Index,
query).from(from).size(size)).documentAs[GitHubRepo]
```

Листинг 4.2: Функција *search*

Ова демонстративна апликација показује како се *ZIO Elasticsearch* може употребити као потпуно типски безбедан и реактиван слој за рад са *Elasticsearch*-ом, без директног писања *Elasticsearch DSL*-а.
Апликација је свесно поједностављена, али и као таква добро илуструје пуни животни циклус рада са *Elasticsearch*-ом: од креирања индекса, преко уписивања и ажурирања података, до нарпедних претрага са разним критеријумима. Оваква архитекрура се може директно применити и на реалне пројекте, било да је реч о претраживању садржаја или аналитичким апликацијама.

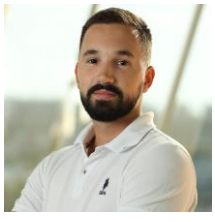
5.ЗАКЉУЧАК

У овом раду приказана је израда библиотеке *ZIO Elasticsearch*, чији је циљ поједностављено и функционално интегрисање *Elasticsearch* претраживачког система са *ZIO* екосистемом. Полазећи од потребе да се корисницима омогући елегантан и типски безбедан начин рада са *Elasticsearch*-ом, развијено је решење које комбинује снагу *ZIO* ефектног модела са богатим могућностима претраге, агрегирања и стримовања података.
Током рада најпре су размотрени кључни концепти и технологије које чине основу библиотеке, након чега је уследила детаљна анализа имплементације. Практична демонстрација библиотеке потврдила је да се *ZIO Elasticsearch* може ефикасно користити у реалним апликацијама, уз очувану читљивост и предвидивост кода. Због функционалног приступа и подршке за *ZIO* екосистем, библиотека олакшава обраду великих количина података, омогућава једноставно управљање грешкама и обезбеђује бољу тестабилност у поређењу са класичним приступима.
Иако је библиотека већ у употребљивом стању, постоји простор за даље унапређење. Будући развој може обухватити проширивање подршке за будуће верзије *Elasticsearch*-а, додавање још напреднијих типова упита и агрегација, као и израду детаљније документације и туторијала који ће допринети лакшем усвајању од стране шире заједнице.
Закључно, развој библиотеке *ZIO Elasticsearch* представља значајан корак ка интеграцији савремених претраживачких система са функционалним програмирањем у *Scala* програмском језику. Овај рад не само да демонстрира техничке могућности и добијене резултате, већ поставља и темеље за будућа истраживања и практичну примену у домену обраде и анализе великих података.

6.ЛИТЕРАТУРА

[1] https://www.elastic.co/ (Последњи приступ Октобар 2025)
[2] https://lucene.apache.org/ (Последњи приступ Октобар 2025)
[3] Paul Chiusano, Runar Bjarnason: “Functional Programming in Scala”
[4] Martin Odersky: “Programming in Scala”
[5] John Hughes: “Why Functional Programming Matters”
[6] https://zio.dev/ (Последњи приступ Октобар 2025)
[7] John A. De Goes, Adam Fraser, Milad Khajavi: “ZIONOMICON”
[8]https://codingexplained.com/coding/elasticsearch/understanding-the-inverted-index-in-elasticsearch/ (Последњи приступ Октобар 2025)
[9] https://www.apache.org/ (Последњи приступ Октобар 2025)
[10]https://www.elastic.co/docs/reference/elasticsearch/rest-apis/paginate-search-results#scroll-search-context/ (Последњи приступ Октобар 2025)
[11]https://www.elastic.co/docs/reference/elasticsearch/rest-apis/paginate-search-results#search-after/ (Последњи приступ Октобар 2025)
[12] https://www.geeksforgeeks.org/software-engineering/separation-of-concerns-soc/ (Последњи приступ Октобар 2025)
[13] https://zio.dev/reference/di/ (Последњи приступ Октобар 2025)
[14] https://zio.dev/reference/contextual/zlayer/ (Последњи приступ Октобар 2025)
[15] https://zio.dev/reference/stream/zstream/ (Последњи приступ Октобар 2025)

Кратка биографија:



Димитрије Булаја рођен је у Зрењанину 1998. године. Дипломирао је на Факултету техничких наука 2021. године и тада уписује мастер академске студије на истом факултету.
Контакт: dbulaja98@gmail.com