Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.42:004.738.12
DOI: https://doi.org/10.24867/10BE04Dragoljevic

РАЗВОЈ СИСТЕМА ЗА ПРЕТРАГУ НАУЧНИХ РАДОВА УПОТРЕБОМ GraphQL–А И ПРОГРАМСКОГ ЈЕЗИКА СКАЛА

DEVELOPMENT OF SCIENTIFIC PAPERS SEARCH ENGINE USING GraphQL AND SCALA PROGRAMMING LANGUAGE

Смиљана Драгољевић, Милан Видаковић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај: *Задатак рада представља развој система за претрагу научних радова. Приликом имплементације серверског дела система употребљен је језик за комуникацију са сервером, GraphQL, а када су у питању програмски језици, систем је реализован употребом програмског језика Scala. Клијентски део система представља апликацију имплементирану помоћу React-a, JavaScript библиотеке. Спецификација система одрађена је употребом UML dijagrama.*
Кључне речи: *Scala, GraphQL, MongoDB, Elasticsearch, React*
Abstract: *The aim of this paper is to develop a search engine for scientific papers. For the implementation of the system's server part GraphQL was used. When it comes to programming languages, the system was implemented using the programming language Scala. The client application was implemented as a React application where React is a JavaScript library. The specification of the system was done using UML diagrams.*

Keywords: *Scala, GraphQL, MongoDB, Elasticsearch, React*

1. УВОД

У раду ће детаљно бити појашњен један од језика који комбинује објектно-оријентисано и функционално програмирање, а то је Скала. Скала ће се демонстрирати на примеру имплементирања веб апликације која представља систем за претрагу научних радова. Детаљан опис ове апликације следи на крају рада. Још једна област која ће бити темељно обрађена јесте једна од алтернатива REST-u (Representational state transfer), а то је *GraphQL*.

Информације о корисницима и радовима, као и *pdf* рада чувају се у *MongoDB* бази података, док је претрага рада по тексту и филтрација резултата омогућена употребом *Elasticsearch*-а, сервера који служи за претрагу и анализу података.

Клијентски део апликације урађен је као *React* апликација, где *React* представља једну од водећих *JavaScript* библиотека за изградњу корисничког интерфејса.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био проф. др Милан Видаковић.

2. ТЕХНОЛОГИЈЕ

У овом поглављу фокус је дат пре свега Скали и *GraphQL*-у, а након тога следи осврт на остале технологије помоћу којих је реализован поменути систем за претрагу научних радова. Како Скала поред подршке за објектно-оријентисано има и пуну подршку за функционално програмирање, дат је увод у функционално програмирање и поменути су основни појмови које ова парадигма уводи.

Функционално програмирање

Функционално програмирање спада у декларативно програмирање, где се акценат даје ономе што програм треба да обави, а не начину на који се неки задатак обавља. Основни појмови функционалног програмирања јесу функције вишег реда, непроменљиви подаци и чисте функције. Функција се посматра као и сваки други тип података. Уводи се појам функције вишег реда, тј. функције којој је могуће проследити функцију као параметар и вратити функцију као повратну вредност. Такође, могуће је варијабли доделити дефиницију функције. Чисте функције су функције које примају улазне параметре на основу којих израчунавају излаз и за исте улазне податке дају исту повратну вредност.

Скала

Скала је статички типизирана комбинација објектно-оријентисаног и функционалног програмирања [1]. Мартин Одески је почео са развојем Скале 2001. године на EPFL-u (École Polytechnique Fédérale de Lausanne), а прва верзија је званично објављена 2004. године. Скала код се компајлира у Јава бајт код који је углавном подједнако ефикасан као и Јава код [2]. Извршава се на Јава виртуелној машини, што пружа могућност коришћења Јава библиотека у Скала коду. Ово је чист објектно-оријентисани језик који сваки тип података, чак и оне који се сматрају прмитивним у другим језицима, посматра као објекат. Иако је у Скали могуће имплементирати апликацију коришћењем чисте објектно-оријентисане нотације, овај језик нуди доста предности уколико се употребљавају функционални концепти које он уводи. Једна од могућности коју нуди и подстиче функционалност Скале јесте рад са непроменљивим подацима (листинг 1).

```
val fruits = List("apple", "banana")
val fruitsUpperCase = fruits.foreach(
  fruit => fruit.toUpperCase())
```

Листинг 1 – Пример трансформације објекта

Поред овога, свака функција у Скали представља вредност те је могуће дефинисати функције вишег реда и угњеждавати функције.
Када се говори о Скали битно је поменути на који начин овај језик решава проблем паралелизације кода. Уводи се библиотека за подршку *Actor* модела. *Actor* је ентитет који комуницира са другим *Actor* -има путем порука [3].

Play framework

Play framework је радно окружење имплементирано у Скали и употребљава се за креирање модерних веб апликација. Користи се за израду кода који се компајлира у Јава бајт код, тј. Јава и Скала кода. *Play* троши минимално ресурса и пружа високо скалабилне апликације [4]. Архитектура овог радног окружења се заснива на MVC (*model-view-controller*) моделу што га чини једноставним и лаким за употребу.

GraphQL

GraphQL је језик за комуникацију са API-јем који нуди ефикасан приступ креирања и обраде захтева клијената упућеним серверској страни апликације. GraphQL сервер подржава, између осталог, и рад са Скалом, Јавом, C++-ом, Пајтоном, итд. Комуникација са сервером путем *GraphQL* -а врши се HTTP (*Hypertext Transfer Protocol*) протоколом преко једног endpoint-а, најчешће POST захтевом. *GraphQL* захтев је текст који се шаље серверу где се интерпретира и обрађује, који потом враћа JSON (*JavaScript Object Notation*) одговор [5]. Подржане су све операције CRUD-а (*create, read, update, delete*). На листингу 2 дат је пример *GraphQL* упита који као одговор очекује наведена поља, и сам одговор са сервера.

```
{
  person {
    name
    age
  }
}
{
  "data": {
    "person": {
      "name": "Anna",
      "age": 25
    }
  }
}
```

Листинг 2 – Пример *GraphQL* упита и одговора

Како би се одвијала комуникација са API-јем, неопходно је дефинисати шему која одређује могуће захтеве као и одговоре сервера на те захтеве. Свако поље шеме има свој назив и тип. *GraphQL* поља могу имати и придружене аргументе. За свако поље је неопходно специфицирати функцију која одређује вредност тог поља наког извршавања упита на серверу. Уколико вредност поља није скаларна, потребно је дефинисати функције за његова поља, све док се не достигну скаларне вредности. Корен *GraphQL* захтева јесте поље *schema*, које даље може да садржи поља типа *Query* и *Mutation*. *Query* означава операције добављања података, без споредних ефеката, док се *Mutation* користи за креирање, модификацију и брисање података. Листинг 3 приказује пример *Query* захтева.

```
query PersonNameAndAge {
  person(id: 1) {
    name
    age
  }
}
```

Листинг 3 – Пример *Query* захтева

Када је дефинисана шема, могуће је валидирати *GraphQL* упит у односу на ту шему. Неке од могућих грешака јесу синтаксна, затим ненавођење жељених поља у склопу поља чији тип није скаларна вредност или навођење поља унутар поља чија вредност јесте скаларна. Уколико је шема валидна, *GraphQL* сервер извршава упит и враћа резултат чија структура наликује упиту, типично у JSON формату [6].
Једна од највећих разлика између *GraphQL* -а и REST архитектонског стила јесте у начину приступа подацима. REST стил налаже да се сваки тип података представи као посебан ресурс и као такав шаље клијенту на захтев. Међутим, у доста случајева се јавља потреба за добављањем више типова података одједном или комбинацијом разних података. *GraphQL* нуди ефикаснији начин приступа ресурсима јер је могуће дефинисати *GraphQL* шему на такав начин да су сви подаци система повезани, те је могуће добавити жељене податке једним захтевом. У случајевима када клијенту нису потребне све ове информације, *GraphQL* има предност у односу на REST, јер се унутар захтева дефинишу жељени подаци које клијент добавља за своје потребе, те је могуће изоставити непотребне вредности.
Имплементација *GraphQL* -а у Скали омогућена је библиотеком *Sangria*. Ова библиотека у потпуности подржава дефинисање шеме са свим типовима које нуди *GraphQL*.

***MongoDB* база података**

MongoDB је дистрибуирана, објектно-оријентисана база података која податке чува у JSON формату. Овакав начин складиштења уводи додатну флексибилност приликом коришћења *MongoDB* јер структура података може да варира од документа до документа. Документи су организовани унутар колекција које углавном чувају документе сличне намене. Ову базу података је могуће користити и као систем датотека. *GridFS* је *Mongo* спецификација за складиштење и добављање већих датотека као што су слике, аудио и видео датотеке, итд [7].

Elasticsearch

Elasticsearch је брз и дистрибуирани сервер за претрагу и анализу разних типова података од безбедносних, преко претраге веб страница до претраге геопросторних података. Такође се користи за одређивање метрике система и складиштење и анализу лог фајлова. Подаци се групишу унутар индекса, где индекс представља сет међусобно повезаних докумената у JSON формату.

React

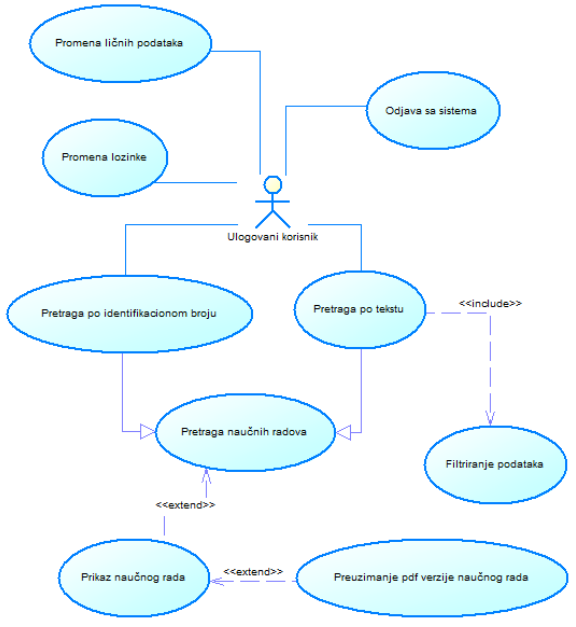
React је *JavaScript* библиотека која омогућава једноставно креирање веб и мобилних апликација. *React* води рачуна само о исцртавању веб страница док је за имплементацију осталих жељених функционалности као што су на пример рутирање апликације и

управљање глобалним стањем, потребно импортовати засебне библиотеке. Свака *React* апликација се састоји од компоненти које су налик функцијама које за повратну вредност имају HTML (*Hypertext Markup Language*) елементе. Приликом креирања апликације користи се екстензија *JavaScript*-а, JSX односно *JavaScript* XML. JSX омогућава писање HTML елемената унутар *JavaScript* -а.

3. СПЕЦИФИКАЦИЈА ЗАДАТКА

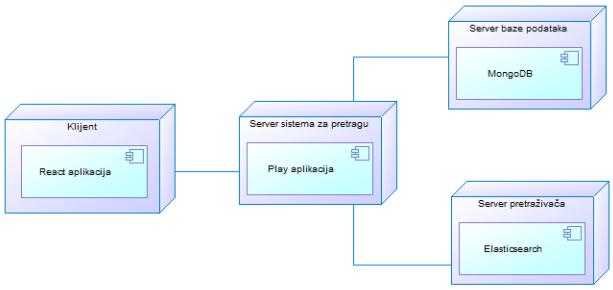
Систем описан у раду представља систем за претрагу научних радова, погодан за употребу унутар једног или више факултета. Информације о раду преузете су из самог рада, тачније из одељка кључна документацијска информација. Такође је доступна и финална верзија рада у *pdf* формату.

Корисник се пријављује на систем користећи свој *email* налог, а уколико није пријављен, нуди му се опција регистрације на систем. Након успешне пријаве, улогованом кориснику се приказују сви радови доступни систему које он потом може претраживати или филтрирати. Једна од претрага понуђена кориснику јесте претрага по тексту рада. Ове радове је могуће додатно филтрирати одабиром године, научне области, ментора, издавача или типа рада. Такође је могућа комбинација поменутих филтера. Уколико је кориснику познат идентификациони број научног рада, може му директно приступити. Пошто је извршена претрага, улоговани корисник има опцију приступа додатним информацијама о раду или по потреби преузимања *pdf*-а рада. Слика 1 приказује дијаграм случајева коришћења улогованог корисника.



Слика 1 – Дијаграм случајева коришћења улогованог корисника

Систем за претрагу научних радова реализован је помоћу неколико компоненти које међусобно комуницирају (слика 2). Серверска апликација представља засебан чвор система који садржи једну *Play* апликацију. Ова апликација комуницира са сервером базе података, конкретно *MongoDB* базе, који представља још једну компоненту система у којој се чувају све информације о раду заједно са његовом *pdf* верзијом. Поред базе података, серверска апликација има везу са *Elasticsearch*-ом чији је сервер такође реализован као засебна компонента која складишти само информације неопходне за претрагу као што су цео текст рада и подаци који се користе приликом филтрирања. Последња компонента јесте клијентска, *React* апликација која комуницира са *Play* апликацијом.



Слика 2 – *Deployment* дијаграм

4. ИМПЛЕМЕНТАЦИЈА

Систем за претрагу научних радова реализован је као вишеслојна апликација која се састоји од слоја података, слоја пословне логике и презентационог слоја. У наставку поглавља следи опис имплементације сва три слоја употребом технологија описаних у поглављу 3.

Складиште података

За реализацију слоја података, једно од коришћених складишта јесте *MongoDB* база података. Састоји се од две колекције које садрже податке о свим научним радовима система и податке о регистрованим корисницима система.

Подаци из *MongoDB* базе података користе се само приликом приказа детаљних информација о научном раду, док се за претрагу рада користе подаци из *Elasticsearch*-а. Апликација користи један индекс, *scientific-paper*, који индексира поља:

- *fulltext* - цео текст рада, употребљен приликом претраге рада по тексту,
- поља за која се формирају на основу филтера и
- поља за приказ радова након извршене претраге.

Серверски део система

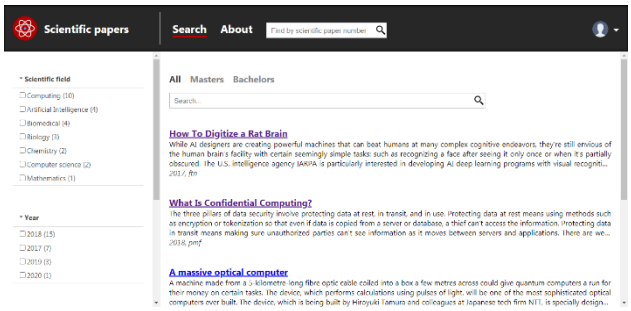
Серверски део система састоји се од једне апликације имплементиране употребом Скале и *Play Framework*-а. Комуникација са овим слојем извршена је употребом језика за комуникацију, *GraphQL*-а, тачније библиотеком која га имплементира у Скали, *Sangria*.

GraphQL шема дефинише поља која представљају операције за претрагу радова, пријаву или регистрацију на систем, промену података о тренутно улогованом кориснику као и промену лозинке уколико ју је корисник заборавио. Креиран је контекстни објекат који садржи методе за израчунавање

сваког од ових поља. Манипулација научним радовима и манипулација корисницима раздвојене су унутар засебних класа, и контекстни објекат садржи објекте ових класа, чијим се индиректним позивом израчунавају вредности *GraphQL* поља.
Приликом имплементације рада са *MongoDB*-ом искоришћен је додатак за *Play*, *ReactiveMongo*. Класа за рад са овом базом садржи све методе неопходне за рад са научним радовима и корисницима. Сервис за рад са *Elasticsearch*-ом имплементиран је употребом клијента *elastic4s*.
Добављање *pdf* формата рада измештено је у засебан контролер, независно од *GraphQL*-а. Рад се проналази на основу идентификационог броја и враћа се у виду *pdf*-а клијенту.

Клијентски део система

Клијентски део система реализован је као *React* апликација, док је комуникација путем *GraphQL*-а имплементирана употребом *Apollo* клијента, *GraphQL* клијента који се, између осталог, интегрише са *React* апликацијама.
Након успешне пријаве на систем, апликација аутоматски шаље захтев за претрагу свих научних радова, без филтрирања. Како је пагинација подржана у систему, клијенту се допрема првих 10 радова, колико је подешено (слика 3).



Слика 3 – Приказ радова након успешне пријаве на систем

Након овога, корисник може одабиром филтера са леве стране приказа филтрирати резултате или, уносом кључне речи у поље предвиђено за то сузити опсег претраге.
Приликом имплементације претраге, креирано је неколико компоненти од којих једна чува стање апликације као што су одабрани филтери, текст за претрагу или жељена страна резултата претраге. Поред овога у њој су дефинисане основне методе за руковање елементима претраге које су као такве, прослеђене одговарајућим компонентама за приказ филтера као и приказ резултата претраге. Приликом исцртавања ове компоненте, као и приликом измене њеног стања кога чине параметри претраге, креира се *GraphQL* захтев, којем се прослеђују поменути параметри, те се овакав захтев шаље серверу. За креирање и слање захтева задужен је *Apollo* клијент.

5. ЗАКЉУЧАК

Систем за претрагу научних радова развијен је обухватајући сет основних функционалности неопходних брзом приступу раду. Даља проширивост система подразумевала би прилагођавање система потребама разних научних установа и корисника. Савремене технологије употребљене приликом имплементације би омогућиле његову једноставну и ефикасну надоградњу, као и одржавање.
Иако и даље коришћен већином у научне сврхе, програмски језик Скала све више и више продире у израду система за свакодневну употребу те на тај начин уводи функционално програмирање у индустријски свет. Такође, језик за комуникацију *GraphQL*, проналази своје место у већим системима како би се корисници што ефикасније прилагодили услугама које нуди сервер.

6. ЛИТЕРАТУРА

[1] Programming in Scala, Martin Odersky, Lex Spoon, Bill Venners, 2007, 2008
[2] A Brief History of Scala, Martin Odersky, June 9, 2006
[3] akka-actor, https://www.javatpoint.com/akka-actor
[4] Play Framework, https://www.playframework.com
[5] GraphQL, https://engineering.fb.com/core-data/graphql-a-data-query-language
[6] GraphQL, https://graphql.org
[7] MongoDB, https://www.tutorialspoint.com/mongodb

Кратка биографија:

Смиљана Драгољевић рођена је 08.11.1995. године у Новом Саду, где је и завршила основну школу „Жарко Зрењанин“. Након тога у истом граду уписује гимназију „Јован Јовановић Змај“ коју завршава 2014. године. Исте године уписује Факултет техничких наука, Универзитета у Новом Саду на смеру Рачунарство и аутоматика. На трећој години студија опредељује се за смер Примењене рачунарске науке и информатика. Дипломирала је 2018. године те је уписала мастер академске студије на истом факултету, смер Рачунарство и аутоматика, модул Електронско пословање. Положила је све испите предвиђене планом и програмом.

Милан Видаковић рођен је у Новом Саду 1971. године. На Факултету техничких наука у Новом Саду завршио је докторске студије 2003. године. На истом факултету је 2014. године изабран за редовног професора из области *Примењене рачунарске науке и информатика*.