Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matevž Žerovec

Spletna aplikacija za podporo učenju stare grščine

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: viš. pred. dr. Alenka Kavčič
Somentor: izr. prof. dr. Matija Marolt

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Spletna aplikacija za podporo učenju stare grščine

Tematika naloge:

V okviru diplomske naloge zasnujte in izdelajte spletno aplikacijo za pomoč pri učenju stare grščine. Aplikacija naj omogoča prikaz in pregledovanje zbranih starogrških besedil, pa tudi brskanje in učinkovito iskanje po besedilih. Iskanje naj omogoča tako iskanje posameznih besed kot tudi le določenih črk ali delov besed. Za shranjevanje tekstovnih podatkov izberite ustrezno podatkovno bazo, ki omogoča učinkovito iskanje po besedilih. Aplikacija naj bo uporabniku prijazna, intuitivna in enostavna za uporabo. Pri njeni izdelavi uporabite sodobne spletne tehnologije in orodja.

Za pomoč pri izdelavi diplomskega dela se zahvaljujem viš. pred. dr. Alenki Kavčič in izr. prof. dr. Matiji Maroltu. Zahvaljujem se tudi svoji družini, prijateljem in vsem, ki so mi na poti do diplome stali ob strani.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
AJAX	Asynchronous JavaScript and XML	asinhroni JavaScript in XML
API	Application Programming Interface	aplikacijski programski vmesnik
CSS	Cascading Style Sheets	kaskadne stilske podloge
JSON	JavaScript Object Notation	objektna notacija JavaScript
MVC	Model-View-Controller	model-pogled-krmilnik
POCO	Plain Old C# Object	Preprost razred C#
REST	Representational State Transfer	arhitektura za izmenjavo podatkov med spletnimi storitvami
XML	Extensible Markup Language	razširljivi označevalni jezik

Povzetek

Naslov: Spletna aplikacija za podporo učenju stare grščine

Avtor: Matevž Žerovec

Dandanes vsebuje vse večji del pedagoškega procesa takšno ali drugačno pomoč multimedijskih vsebin. Cilj tega diplomskega dela je bil, da se razvije enega od takih pripomočkov, torej sistem, ki bo v pomoč pri učenju antične grščine. Rezultat je aplikacija, ki omogoča branje praktično vseh starogrških besedil. V ta namen je bilo potrebno v prvi fazi poiskati ustrezne vire besedil in jih predelati v takšno obliko, da jih je bilo možno vstaviti v podatkovno bazo. Ker je bila ena od zahtev ta, da se nad celotno zbirko besedil med drugim omogoči iskanje po besedah in črkah, smo se odločili, da bomo za zaledni sistem uporabili sistem Elasticsearch, saj je med drugim namenjen poizvedbam, ki se vršijo nad celotnimi besedili. Ostali del aplikacije je realiziran v programskem jeziku C# in ogrodju .NET MVC 5.

Ključne besede: antična grščina, spletna aplikacija, iskanje po besedilih, Elasticsearch, knjižnica.

Abstract

Title: Web application for supporting learning of Ancient Greek

Author: Matevž Žerovec

In today's world more and more of the teaching process involves some kind of multimedia help. The goal of this thesis was to develop a system that is going to help with learning ancient Greek. The result is an application, that allows you to read practically any of the ancient Greek texts. For that we first needed to find appropriate source of such texts and process them in such a way that they would fit into our database system. Because one of the demands for the application was for it to be able to search the whole corpus of texts by words or characters, we opted for Elasticsearch for our back end system. That is because Elasticsearch is, among other things, intended to be used for full text searches. The rest of the application was built in C# and .NET MVC 5 framework.

Keywords: ancient Greek, web application, full text search, Elasticsearch, library.

Poglavje 1

Uvod

Informacijska tehnologija nam v današnjem času lahko pomaga pri nalogah, za katere smo nekdaj porabili veliko več časa, truda in energije. Na primer iskanje po slovarju je včasih zahtevalo listanje, koncentracijo in pozornost, danes pa samo nekaj klikov. Za iskanje knjig v knjižnici je bilo potrebno poznati sistem shranjevanja, morda smo potrebovali celo pomoč knjižničarja. Predstavljajmo si, kako zamudno bi bilo grajenje seznama najpogostejših besed neke zbirke knjig ali pa iskanje vseh pojavitev določene besede v zbirki. Z razvojem informacijske tehnologije lahko marsikatera od teh opravil postanejo stvar samo nekaj klikov uporabnika.

Podobne probleme želi za tiste, ki se učijo ali poučujejo staro grščino, na enem mestu rešiti aplikacija, ki je tema tega diplomskega dela. Pobuda oziroma želja za tako aplikacijo je prišla s strani profesorjev za staro grščino na Filozofski fakulteti Univerze v Ljubljani. Ideja je bila, da se zgradi aplikacijo, ki na prvem mestu omogoča vpogled v kar se da veliko število starogrških tekstov. Nadaljnje naj bi aplikacija omogočala filtriranje tekstov po določenih filtrih, kot sta na primer avtor dela ali ime dela. Prav tako naj bi v aplikaciji imeli možnost iskanja posameznih besed v zbirki tekstov in iskanja določenih črk in delov besed.

V diplomskem delu je predstavljen postopek implementacije teh idej v obliki spletne aplikacije. Pred začetkom razvoja smo morali poiskati ustrezen

vir besedil, saj tega na začetku razvoja nismo imeli. Pridobljena besedila je bilo potem potrebno spraviti v obliko, ki je ustrezala zalednemu sistemu in jih vanj tudi shraniti. Na to je prišel na vrsto tudi razvoj aplikacije. Aplikacija je zgrajena v okolju ASP.NET MVC 5 in programskem jeziku C#, za zaledni del pa smo uporabili iskalni pogon Elasticsearch. Odločitev za Elasticsearch ni bila težka, saj je ta namenjen reševanju problemov, s katerimi se sooča aplikacija, torej iskanju besed oziroma črk po veliki zbirki tekstov.

Poglavje 2

Funkcionalnosti in primerjava z obstoječimi aplikacijami

Branje starogrških tekstov in iskanje po njih sta bili dve izmed osnovnih zahtev za aplikacijo. Ta del funkcionalnosti na spletu pokriva že več aplikacij oziroma digitalnih knjižnic. V nadaljevanju bomo predstavili zahtevane funkcionalnosti aplikacije in jih primerjali z obstoječimi sistemi.

2.1 Želene funkcionalnosti aplikacije

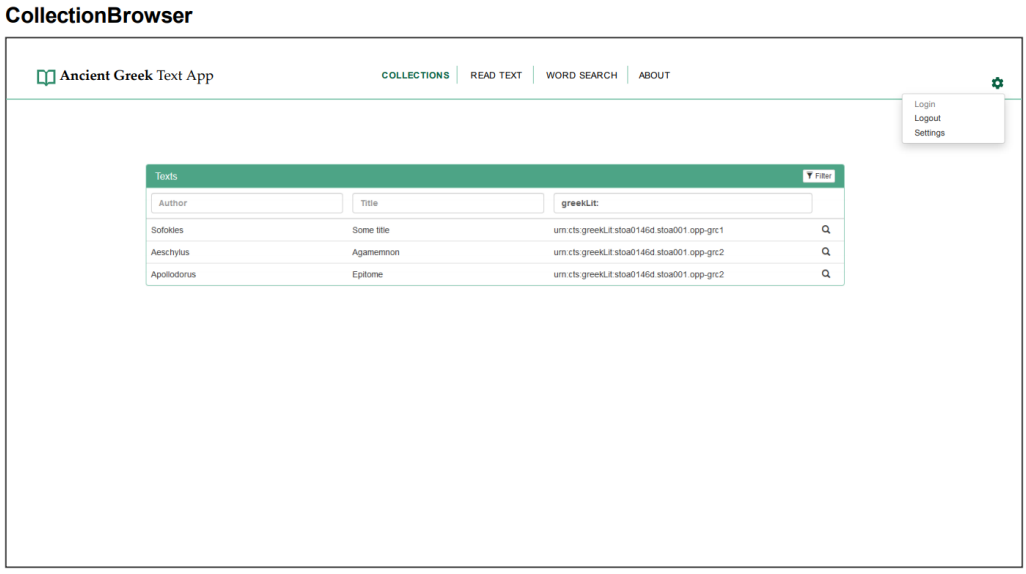
Potreba po aplikaciji, ki je nastala v sklopu diplomskega dela, je nastala na Filozofski fakulteti v Ljubljani na oddelku za klasično filologijo. Obstoječi pripomočki več niso bili primerni za uporabo, prav tako pa niso bili odprtokodni, tako da nadgradnja obstoječega sistema ni prišla v poštev.

Odločitev je torej bila, da se zgradi nov sistem, ki bo implementiral želje in potrebe, ki so nastale med poučevanjem stare grščine in raziskovanjem tekstov. Tako smo na uvodnem sestanku okvirno definirali funkcionalnosti, ki bi jih podpirala nova aplikacija:

- Iskanje tekstov po naslovu in avtorju;
- pregledovanje posameznega teksta in razlikovanje med prozo in poezijo;

- prikaz najpogostejših besed v zbirki besedil;
- iskanje besed v zbirki besedil in navigiranje na lokacijo besede v besedilu;
- iskanje besed, ki so sestavljene iz določenega nabora znakov.

Glede na seznam zahtev smo pripravili žični model, ki je bil v fazi razvoja aplikacije v pomoč pri načrtovanju. Na spodnji sliki 2.1 je prikazan rezultat izdelave žičnega modela za prvo masko aplikacije.



Slika 2.1: Žični model vstopne maske aplikacije.

2.2 Iterativno nadgrajevanje aplikacije

Skozi celotno fazo razvoja aplikacije sta sledila še dva sestanka vseh udeležencev, kjer smo pregledali narejeno delo, obravnavali opazke testiranja in določili nadgradnje. Tako smo aplikaciji skozi razvoj določili še dodatne funkcije. Bralnik teksta je tako dobil funkcijo, da se po dvokliku besede v novem oknu

odpre iskanje te besede na slovarskem spletnem portalu Termania [1]. Skozi iteracije je dobila aplikacija tudi tri načine iskanja. Najbolj osnovni način je iskanje po besedah. Uporabnik vpiše besedo, iskalnik pa vrne vsa besedila, ki vsebujejo to besedo. Po izbiri besedila se odpre bralnik teksta z dodatkom, ki uporabnika vodi po zadetkih iskane besede v besedilu, kot je vidno na desni strani spodnje slike 2.2.

Slika 2.2: Prikaz zadetkov v besedilu.

Vir besedil, ki ga uporablja aplikacija, nam omogoča tudi iskanje po lemah. Leme so osnovne oblike besed. Za samostalnike in pridevnike to pomeni prva oseba ednine, za glagole nedoločna oblika in tako naprej. Tako vsi načini iskanja v aplikaciji ponujajo možnost, da se iskanje vrši nad lemami besed. Prav tako nam dvoklik na besedo v bralniku teksta odpre iskanje po slovarju v osnovni obliki besede. To je tudi ena od glavnih prednosti aplikacije pred večino podobnih aplikacij na trgu, katere bomo v nadaljevanju primerjal z našim končnim izdelkom.

2.3 Primerjava z obstoječimi aplikacijami

Na spletu obstaja veliko aplikacij, ki ponujajo branje preslikanih starogrških dokumentov, kar po navadi pomeni, da ne ponujajo iskanja po teh dokumentih. V tem poglavju se bomo osredotočili samo na aplikacije, ki ponujajo dokumente v digitalni tekstovni obliki, saj take aplikacije omogočajo tudi takšno ali drugačno iskanje po dokumentih.

Projekt Gutenberg [2] je zbirka več kot petdeset tisoč knjig v digitalni tekstovni obliki. Med njimi so tudi teksti starogrških piscev. Kot naša aplikacija tudi ta omogoča iskanje po avtorju, naslovu, besedah in po delih besed. Razlikuje se v tem, da ne omogoča prostega iskanja po črkah v besedi ne glede na vrstni red in iskanja po lemah.

Precej priljubljena je digitalna knjižnica Perseus [3]. Večina tekstov v tem sistemu izvira iz antike, tako grške kot rimske, prav tako pa nudi tudi angleške prevode teh tekstov. Iskalnik te knjižnice omogoča iskanje po besedah, delih besed ter avtorju in naslovu. Uporabna je funkcija, ki za vneseno angleško besedo v tekstih poišče prevode te besede. Na primer iskanje po „spirit“ bo našlo besede, kot so: „animus“, „genius“, „spiritus“. Ponuja tudi zanimivo iskanje po referencah za osebami, kraji in časi, nima pa možnosti iskanja po lemah in prostega iskanja besed po vnesenih črkah.

Najbolj funkcionalno bogata od teh treh sistemov je zagotovo digitalna knjižnica TLG [4], ki je raziskovalni projekt Univerze v Kaliforniji. Njen cilj je zgraditi obširno digitalno knjižnico tekstov iz antične Grčije. Njen iskalnik ponuja vse od zgornjih dveh, poleg tega pa tudi časovno in krajevno umestitev tekstov. Enako kot naš sistem omogoča tudi iskanje po lemah, prav tako pa prikaže tudi lokacije najdenih rezultatov v tekstih. Prednost naše aplikacije v primerjavi z digitalno knjižnico TLG pa je neposredna povezava na slovenski slovar v bralniku teksta.

Poglavje 3

Uporabljene tehnologije

Pred vsakim začetim projektom si razvijalec programske opreme ali ekipa postavi vprašanje, v kateri tehnologiji oziroma ogrodju se bo razvijal projekt. Odločitev je težka, saj je po navadi precej trajna. Prepisovanje stare kode v novih ogrodji je namreč drago oziroma časovno potratno. Priporočljivo je, da se uporabi čim novejše uveljavljeno ogrodje, ki ekipi oziroma posamezniku ustreza. Tako si zagotovimo podporo za morebitne kasnejše nadgradnje in uporabo najnovejših standardov, uveljavljenih v industriji razvoja programske opreme.

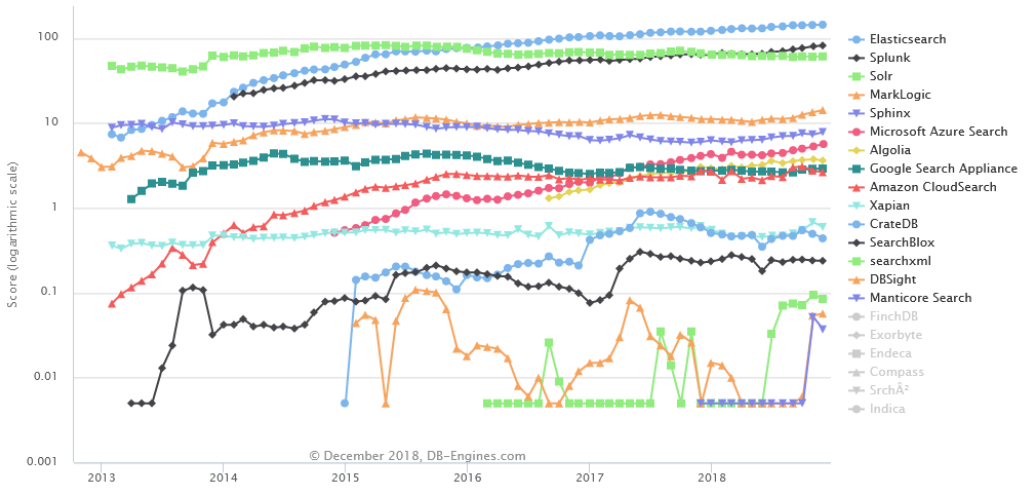
Pri našem delu se je bilo potrebno najprej odločiti, kateri pogon bo podpiral zaledni del in na katerem bo zgrajena spletna aplikacija. Za zaledni del smo se odločili za pogon Elasticsearch. Elasticsearch med drugim namreč ponuja hitro iskanje po besedilih. To pomeni, da omogoča na primer iskanje določene besede na veliki količini teksta v skoraj realnem času, kar pa je ena od ključnih funkcionalnosti naše aplikacije.

Za razvoj spletnega dela aplikacije pa smo se odločili za ogrodje ASP.NET MVC 5. Ogrodje v času odločitve sicer ni bilo več najnovejše, vendar smo se vseeno odločil zanj, saj sem ogrodje poznal, kar pa je očitna prednost pred ogrodji, katerih bi se moral še naučiti.

3.1 Elasticsearch

Elasticsearch [5] je odprtokodni iskalni strežnik, ki je zgrajen na zelo močni, prav tako odprtokodni, javanski knjižnici Lucene [6]. Elasticsearch deluje preko REST HTTP protokola, kar pomeni, da je dostopen praktično vsem programskim jezikom. Za hrambo dokumentov v indeksu in komunikacijo preko RESTful spletnega vmesnika uporablja format JSON. Elasticsearch nudi tudi horizontalno razdeljevanje indeksov preko tako imenovanih črepinj (eng. database shard). To pomeni, da se logična podatkovna baza, oziroma v primeru Elasticsearcha indeks, razdeli na več neodvisnih delov. V velikih sistemih to pomeni lažjo dostopnost indeksov, razdeljevanje bremena in lažje vzdrževanje.

Vse od naštetega vpliva na to, da je Elasticsearch, sodeč po raziskavi DB-Engines [7], ena najbolj priljubljenih odločitev pri izbiri iskalnega strežnika na trgu. Na sliki 3.1 je namreč vidno, da je konec leta 2015 Elasticsearch prehitel še projekt Apache Solr, ki je prav tako zgrajen na knjižnici Lucene.



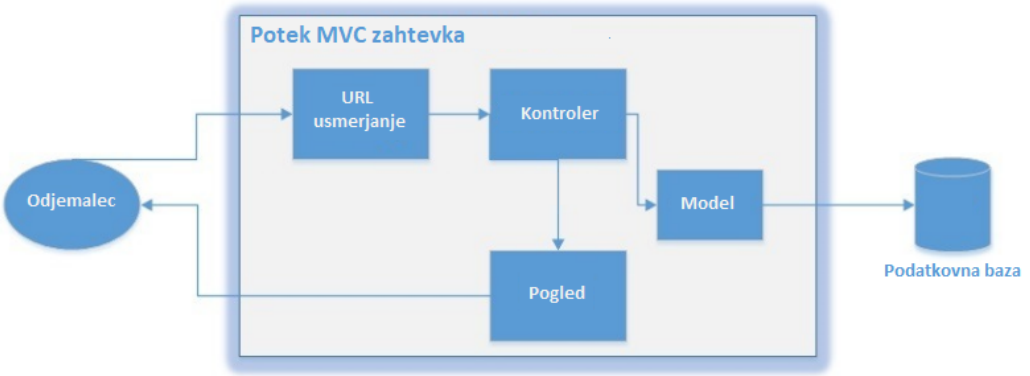
Slika 3.1: Graf trenda priljubljenosti iskalnih strežnikov

3.2 Ogrodje .NET

.NET [8] je dobro poznano Microsoftovo ogrodje za razvoj programske opreme, ki, razen najnovejših verzij (.NET Core), teče večinoma samo na operacijskih sistemih Windows. Programi, napisani v ogrodju .NET, se izvajajo na virtualni napravi Common Language Runtime, ki poskrbi za varnost, krmili z izjemami v izvajanju in upravlja z delovnim spominom. V ogrodju so skozi čas nastajala različna okolja za razvoj. Leta 2013 je izšel tudi ASP.NET MVC 5, v katerem smo razvili spletni del aplikacije.

3.2.1 ASP.NET MVC 5

ASP.NET MVC 5 [9] je ena izmed Microsoftovih odprtokodnih implementacij modela MVC v .NET okolju. MVC arhitekturni model, kot je prikazan na sliki 3.2, se uporablja večinoma pri razvoju spletnih aplikacij. Model skrbi za jasno ločnico med deli programske kode glede na njihovo namembnost.



Slika 3.2: Arhitekturni model MVC

V delu model (angl. Model) arhitekture MVC se nahajajo objekti, ki so v domeni aplikacije. Navadno so ti predstavljeni v preprostih razredih POCO (Plain Old C# Object), ki so zrcalne slike objektov, shranjenih v podatkovni bazi. V našem primeru so se v modelu nahajali razredi C#, ki

so predstavljali strukturo dokumentov JSON, hranjenih v Elasticsearchovem indeksu.

Kontroler (angl. Controller) je del, ki skrbi za odziv na zahtevo HTTP, ki pride na strežnik. Zahteva je pred tem poslana na pravi kontroler skozi usmerjevalnik zahtev. Kontrolerji so v implementaciji .NET MVC-ja posebni statični razredi C#. V njih se neposredno, ali pa posredno (preko klicev na druge razrede ali knjižnice), izvaja poslovna logika aplikacije in komunikacija s podatkovno bazo s pomočjo modela.

Kontroler uporabniku, ki je poslal zahtevo na strežnik, vrne pogled (angl. View), ki je končni rezultat obdelave zahteve na strežniku. V MVC 5 se za gradnjo pogleda uporablja pogon Razor, ki omogoča pisanje hibridnih datotek. V teh datotekah lahko kombiniramo označevalni jezik HTML s kodo C#, s čimer pridobimo dodatno fleksibilnost pri implementaciji.

3.2.2 Elasticsearch.Net in knjižnica NEST

Razvijalci Elasticsearcha so razvili več uradnih odjemalcev za različne programske jezike. Med drugim so bile razvite tudi knjižnice za okolje .NET. To sta nizkonivojski Elasticsearch.Net [10] in visokonivojski NEST [11]. Elasticsearch.Net skrbi za to, da so vse metode REST spletnega vmesnika Elasticsearch API preslikane v okolje .NET, brez da bi to bila ovira, kako programer želi zgraditi zahteve JSON za Elasticsearch API.

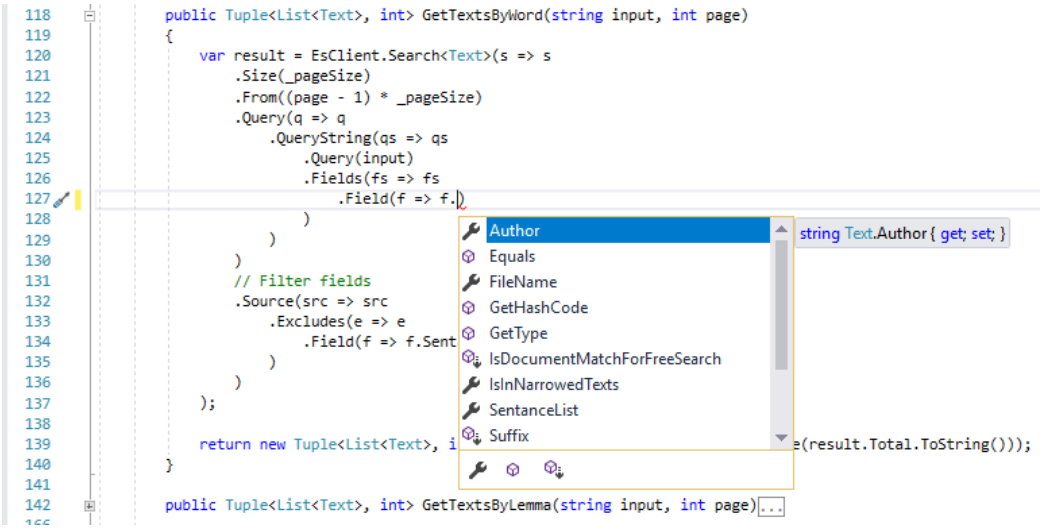
NEST je visokonivojski odjemalec, ki na nižjem nivoju uporablja Elasticsearch.Net. NEST žrtvuje popolno svobodo, ki jo imamo na nižjem nivoju, za to, da pridobimo močno tipiziran (angl. Strongly typed) način klicanja vmesnika Elasticsearch API. To pomeni manj napak pri programiranju in omogočanje urejevalnikom, da nudijo pomoč razvijalcem preko samoizpolnjevanja kode. NEST poskrbi tudi za avtomatično preslikavo med dokumenti v indeksih Elasticsearch in modelih POCO.

3.3 Uporabljena orodja

3.3.1 Visual Studio 2017

Visual Studio 2017 je Microsoftovo razvojno okolje, ki se uporablja za razvoj aplikacij za operacijski sistem Windows, za spletne aplikacije in storitve in tudi mobilne aplikacije. Podpira razvoj v jezikih C#, Visual Basic .NET, F#, C++, C, Javascript, so pa na voljo tudi razširitve za druge programske jezike.

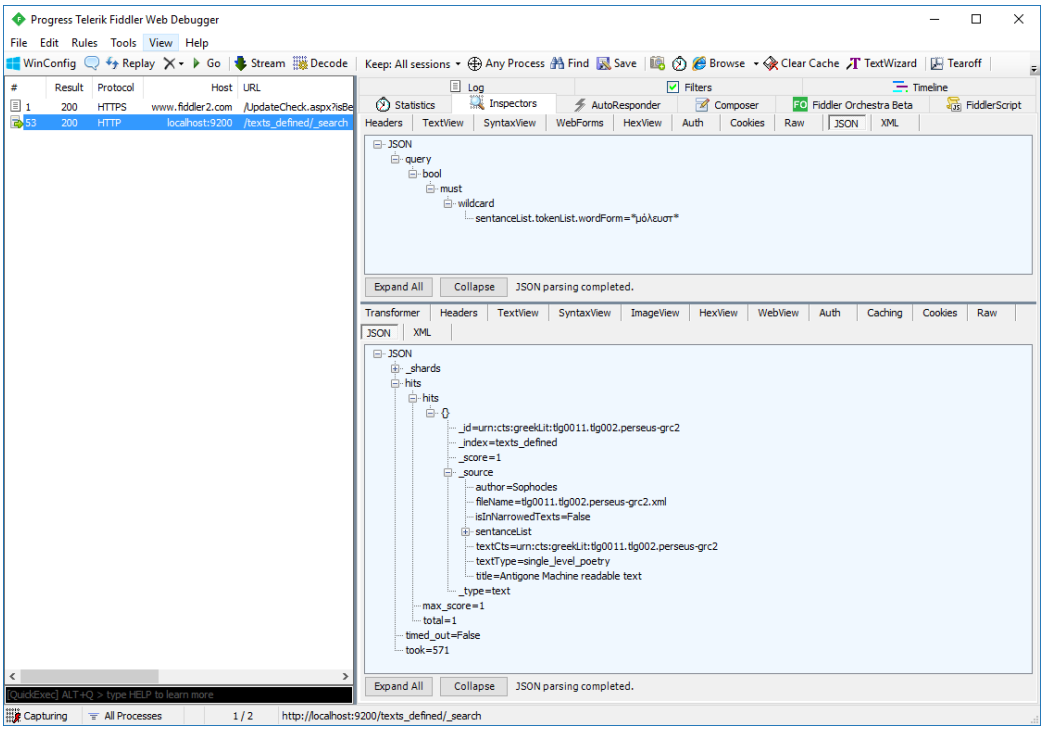
Okolje je zelo bogato s funkcionalnostmi. Omogoča razhroščevanje v času izvajanja z vpogledom v objekte, ki so trenutno v spominu, in celo spreminjanje kode med samim razhroščevanjem. Vpogled v objekte je del IntelliSense [12] pripomočka v Visual Studiu. IntelliSense nudi tudi avtomatično dopolnjevanje kode v realnem času, označitev sintaktičnih napak pred izvajanjem in prikaz namigov za lepšo kodo ter izogib slabim praksam. Avtomatično dopolnjevanje je lepo vidno na spodnji sliki 3.3, kjer IntelliSense ponuja atribute objekta, ki predstavlja dokument v Elasticsearchu.



Slika 3.3: Prikaz delovanja IntelliSense

3.3.2 Fiddler 4

Fiddler [13] je orodje, ki pomaga pri pošiljanju in razhroščevanju zahtev HTTP. Fiddler zajema promet HTTP in ga prikaže uporabniku s pomočjo certificiranega napada s posrednikom [14]. Navadno je zahtev in odgovorov HTTP že na osebnih računalnikih zelo veliko, zato Fiddler ponuja filtriranje odgovorov. Uporabniku omogoča tudi formatiran prikaz odgovorov HTTP, na primer v drevesni strukturi JSON, kar zelo pripomore k berljivosti odgovorov. Na spodnji sliki 3.4 je v zgornjem okviru prikazana strukturirana poizvedba na Elasticsearch, v spodnjem okviru pa je viden strukturiran odgovor.



Slika 3.4: Fiddlerjev prikaz poizvedbe in odgovora

Poglavje 4

Priprava podatkov in indeksa

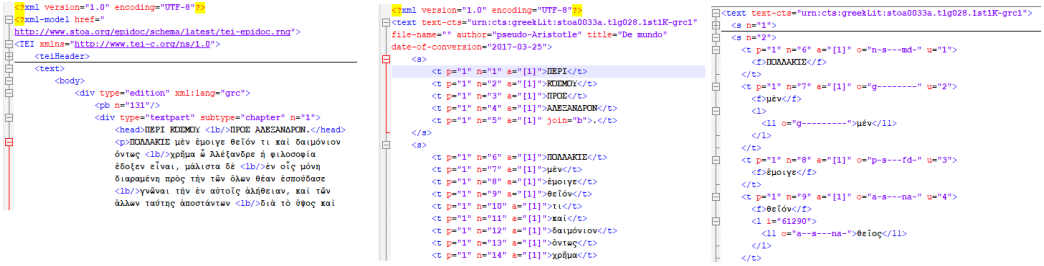
Že takoj ob prevzemu teme za diplomsko delo je bilo jasno, da se bo delo delilo na dva sklopa. To sta razvoj aplikacije in pridobitev virov, katere bo aplikacija uporabljala. Ker je pred razvojem aplikacije potrebno poznati obliko podatkov in uporabljen zaledni sistem za podatke, je najprej prišlo na vrsto iskanje virov. Za tem je bilo potrebno podatke še preoblikovati in jih indeksirati v pripravljen Elasticsearchov indekes. V tem poglavju bom opisal reševanje naštetih problemov.

4.1 Izbira vira podatkov

Pri iskanju vira smo imeli v mislih to, da bo izbira vplivala na težavnost razvoja oziroma da nas lahko omejuje pri implementaciji zastavljenih ciljev. Glede na to, da smo vedeli, da bomo za zaledni sistem uporabili Elasticsearch, bi lahko uporabili enega od mnogih virov, ki ponujajo originalno digitalizirano obliko starogrških tekstov. Elasticsearch je namreč narejen prav za procesiranje in obdelavo velike količine teksta, kar pomeni, da struktura teksta za iskanje po njem ne igra velike vloge.

Ker pa je bila ena od osnovnih zahtev tudi uporabniku prijazno prebiranje teksta, smo se zaradi lažje implementacije zahtev odločili za bolj strukturiran vir podatkov. Na spletu smo tako našli kar nekaj različnih repozitorijev,

ki ponujajo starogrške tekste, ki so na nek način strukturirani v dokumentih XML. Samo razčlenjevanje tekstov je bilo v različnih virih narejeno na različnih nivojih. Tako na primer projekt First1KGreek [15], eden primernih repozitorijev za izbiro vira podatkov, ponuja dokumente, razčlenjene na nivoju odstavkov. Še boljšo razčlenitev ponujata repozitorija CTS Ancient Greek XML [16] in Lemmatized Ancient Greek Texts [17]. Oba sta nadgradnja repozitorija First1KGreek, saj razčlenita besedilo na nivo besede. Zaradi lažjega vpogleda v strukturo besedila pri programiranju smo se odločili za repozitorij Lemmatized Ancient Greek Texts. Ta ima namreč pred repozitorijem CTS Ancient Greek XML še to prednost, da je besedilo tudi lematizirano, kar pomeni, da večina besed s seboj nosi tudi podatek o svoji osnovni obliki – lemi. Na spodnji sliki 4.1 je prikazana oblika dokumentov, ki so bili v naboru za izbiro vira. Zadovoljni smo bili tudi s samim obsegom korpusa repozitorija Lemmatized Ancient Greek Texts, saj je ta vseboval več kot 800 starogrških tekstov, kar je pomenilo, da bo aplikacija bogata tudi z vsebinskega vidika.



Slika 4.1: Primerjava virov – od leve proti desni: First1KGreek, CTS Ancient Greek XML, Lemmatized Ancient Greek Texts

Poleg dobrodošlega dodatka lem so dokumenti XML tega repozitorija vsebovali še metapodatke o svoji strukturi oziroma neke vrste dokumentacijo. Na sliki 4.2 točno vidimo, kako je besedilo razčlenjeno. Vozlišču „text“ sledi „sentence“ in znotraj še „token“ in „wordForm“. V osnovi je bilo torej besedilo razčlenjeno na stavke in besede. Vsaka beseda nosi potem še potencialno

dve lemi iz različnih virov znotraj vozlišča „lemma“. Zaradi sestavljanja teksta v sami aplikaciji je bil najpomembnejši atribut „p“ v vozlišču „token“. Ta je nosil informacijo o tem, v katerem poglavju oziroma kitici, če je šlo za poezijo, se nahaja beseda. Za tem imamo še nekaj atributov, ki opisujejo položaj znotraj poglavja ali kitice in položaj v stavku. Zanimiv je tudi atribut „o“, ki predstavlja zastavice o obliki besede. Torej ali gre za samostalnik, glagol, v katerem času ali spolu je beseda in podobno. Pravila so definirana v repozitoriju Git [17].

```
<text text-cts="urn:cts:greekLit:stoa0121.stoa001.opp-grc1" author="Eutropius" title="Breviarium
historiae romanae" file-name="stoa0121.stoa001.opp-grc1.xml" date-of-conversion="2017-03-29">
  <sentence n="position in the document">
    <token p="passage-level cts urn" n="position in @p" a="nth occurrence in @p" o="Mate tagger POS tag"
    u="position in s element" join="optional: join b(ack) or a(fter) for punctuation marks" tag=
    "optional: selection of editorial tags containing the token in the original document, such as del or
    add">
      <wordForm>word form of the token</wordForm>
      <lemma id="contains the id number of the entry in the database consisting of the set of all word
      forms + morphological analyses of all Greek texts">
        <l1>as many l1 elements as the lemmas found in PerseusUnderPhilologic for
              the relevant word form AND morphological analysis</l1>
        <l2>as many l2 elements as the lemmas found in Morpheus for
              the relevant word form AND morphological analysis</l2>
      </lemma>
    </token>
  </sentence>
  <s n="1">
    <t p="1.1" n="1" a="[1]" o="p-s---fg-" u="1">
      <f>Τῆς</f>
      <l i="123453">
        <l1 o="pa-s---fg-">ὁ</l1>
      </l>
    </t>
```

Slika 4.2: Struktura dokumenta XML repozitorija Lemmatized Ancient Greek Texts

4.2 Predelava virov in indeksiranje

Po tem, ko smo se odločili za vir podatkov, je bilo tega potrebno indeksirati v Elasticsearchu, še pred tem pa namestiti Elasticsearch in ustrezno nastaviti parametre za nemoteno delovanje.

Namestitev Elasticsearcha za operacijski sistem Windows je relativno pre-

prosta, predpogoj za delovanje pa je vsaj Java 8. Elasticsearch bo uporabil tisto verzijo Jave, ki je shranjena v okoljski spremenljivki `JAVA_HOME`. Elasticsearch je najprej potrebno prenesti z uradne spletne strani v obliki .zip. Nato ga odarhiviramo na želeno mesto in potem v mapi bin najdemo elasticsearch.bat. Po zagonu bi se moral Elasticsearch odzivati na vratih 9200. Povezavo lahko testiramo v katerem koli brskalniku s klicem `http://localhost:9200/`. Ta klic bi moral vrniti osnovne podatke o verziji Elasticsearcha.

Ko smo imeli strežnik nameščen, je prišla na vrsto predelava podatkov v tako obliko, da se bodo dokumenti lahko indeksirali v Elasticsearchu. Elasticsearch namreč pričakuje dokumente v obliki JSON, tako je bilo potrebno naše dokumente XML pretvoriti v JSON. Na srečo je za to skrbela uradna knjižnica .Net za Elasticsearch, NEST. Knjižnica namreč v končni fazi omogoča serializacijo razredov C# v dokumente JSON za komunikacijo z Elasticsearchom. Razvijalec mora tako samo poskrbeti, da pripravi ustrezne razrede C#, po tem pa nadaljuje razvoj na visokem, objektnem nivoju. Funkcije knjižnice NEST sprejemajo objekte C#, v ozadju na nižjem nivoju pa izvršijo serializacijo v dokumente JSON. S temi dokumenti potem NEST pokliče Elasticsearch in deserializira morebitni odgovor JSON spet v objekte C#. Tako smo za potrebe indeksiranja ustvarili ločen konzolni program .Net. Sprva je služil samo indeksiranju virov XML, kasneje pa je dobil tudi druge funkcije, ki jih bom opisal kasneje.

Preden smo dokumente indeksirali, smo v konzolni aplikaciji implementirali še razred, ki se je sprehodil čez vse dokumente XML in jim oklestil metapodatke, vidne na sliki 4.2. Informacija o strukturi dokumenta, ki jo je vseboval vsak XML, namreč ni služila nobenemu namenu in je bila samo dodatna prostorska obremenitev. Vse ostale podatke smo obdržali zaradi takojšnje uporabnosti in zaradi možnosti nadgradnje aplikacije. Za samo indeksiranje smo najprej poizkusili uporabiti tako imenovani Bulk klic API. Ta nam omogoča, da hkrati indeksiramo velike količine podatkov, kar je v našem primeru pomenilo 861 dokumentov XML, kar je naneslo malo več kot 2 GB

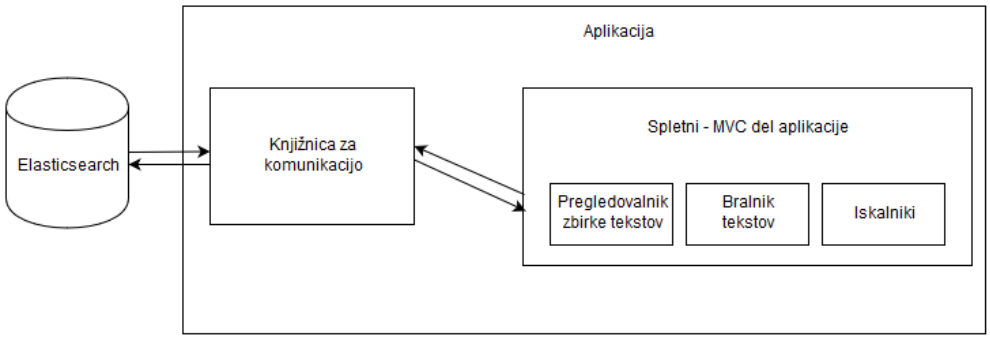
prostora. Tako je naš program za indeksiranje najprej deserializiral vse dokumente XML v razrede C# in nato poskusil z klicem Bulk indeks s pomočjo knjižnice NEST, vendar je strežnik podlegel zaradi pomanjkanja delovnega spomina. Problem je bila Javina implementacija kopice, ki jo uporablja Elasticsearch, saj ta porabi veliko prostora za razvejane razrede. V našem primeru je bilo to še posebej očitno, saj je bila vsaka beseda in njena lema shranjena v svojem končnem vozlišču XML, in kasneje, po deserializaciji, razredu. Zato je bilo potrebno opustiti klic Bulk in dokumente deserializirati ter indeksirati enega za drugim. To je rešilo težave s delovnim spominom, saj je lahko Javin čistilec spomina (angl. garbage collector) sproti ustvaril prostor za procesiranje novih dokumentov. Celo pri indeksiranju posameznih dokumentov smo, pri posebej velikih dokumentih, prišli pri privzeti konfiguraciji Elasticsearcha do težave z pomanjkanjem delovnega spomina. Privzeta nastavitev namreč Javini navidezni napravi (angl. Java Virtual Machine – JVM) dodeli samo 1 GB delovnega spomina, zato smo to vrednost povečali na 8 GB. To je namreč polovica delovnega spomina naprave, na kateri se je projekt razvijal, kar je bilo priporočilo razvijalcev Elasticsearcha. Po tem ko smo vrednosti spremenili, smo imeli uspešno pripravljen indeks Elasticsearch z našimi dokumenti. Celotni korak indeksiranja je bilo potrebno izvesti samo enkrat in ko je bil indeks pripravljen, je bilo iskanje po tekstih hitro in ni zahtevalo take količine delovnega spomina kot indeksiranje.

Kasneje v razvoju smo dodali v Elasticsearch še en indeks. Ta je precej bolj preprost in nosi samo podatke o najpogostejših besedah celotnega korpusa besedil. Ta indeks se ne gradi v tem programu, ampak se ga lahko ponovno zgradi v sami spletni aplikaciji.

Poglavje 5

Razvoj aplikacije

Po uspešno postavljenem indeksu smo bili pripravljeni na gradnjo same aplikacije. Kot rečeno, smo izbrali ogrodje .NET MVC 5. Še pred razvojem smo začrtali tudi arhitekturo aplikacije in jo na najvišjem nivoju ločili na dva dela, kot je vidno na sliki 5.1. Prvi del je klasična razredna knjižnica .NET, ki se prevede v datoteko .dll. Knjižnica skrbi predvsem za komunikacijo z Elasticsearchom. S tem smo ločili podatkovni nivo od poslovnega nivoja, saj drugi del aplikacije, torej sama aplikacija MVC oziroma posamezni kontrolerji znotraj spletnega dela aplikacije, ni več odvisen od Elasticsearcha, ampak za komuniciranje z bazo uporablja zgolj temu namenjeno knjižnico. V nadaljevanju bomo predstavili samo aplikacijo in rezultate razvoja.



Slika 5.1: Arhitektura aplikacije

5.1 Komunikacija z Elasticsearchom

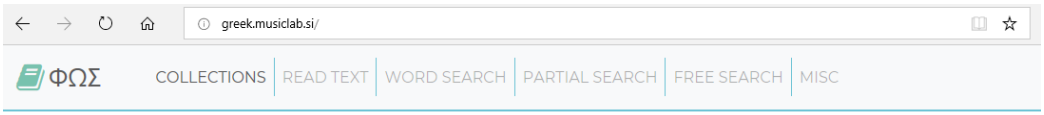
Kot rečeno, je bila za neposredno komuniciranje z Elasticsearchom napisana posebna knjižnica. Podobno kot v programu za pripravo podatkov smo tudi tukaj uporabili knjižnico NEST, kar nam je prihranilo precej odvečnega kodiranja, saj so vsi klici, ki jih ponuja Elasticsearch, implementirani v tej knjižnici .NET. Po priporočilu razvijalcev smo razred, ki predstavlja odjemalca, implementirali kot singleton. To pomeni, da lahko en proces aplikacije ustvari samo eno instanco tega razreda skozi svoj celotni življenjski cikel.

Sama knjižnica za komunikacijo ima ločena odjemalca za indeks z dokumenti in indeks z najpogostejšimi besedami. Za oba indeksa sta zgrajena tudi razreda, ki imata implementirane metode za komunikacijo z Elasticsearchom glede na potrebe spletne aplikacije. Razreda služita kot vmesnik med spletno aplikacijo in Elasticsearchom. Recimo, da uporabnik vnese iskalni niz „Platon“. Kontroler bo poklical poenostavljeno metodo razreda, ki je zadolžen za indeks dokumentov. Metoda sprejme en parameter, ime avtorja in vrne vse dokumente tega avtorja kot domenske objekte C#. V tej poenostavljeni metodi se zgodi klic ustrezne metode knjižnice NEST in obravnava rezultata tega klica. Kontrolerju lahko vrnemo napako ali pa izluščimo domenske

objekte C# iz odgovora in jih podamo naprej.

5.2 Spletni del aplikacije

V spletnem delu aplikacije smo osnovnemu ogrodju MVC, za katerega poskrbi Visual Studio, dodali še predelan stil CSS in Javascriptovo knjižnico jQuery za pomoč pri manipulacij z elementi DOM. Posamezne dele aplikacije, kot je bralnik, različne iskalnike in druge dele, smo razdelili na zavihke oziroma podmenije, kot je vidno na spodnjem zajemu zaslona 5.2. Praviloma ima vsak zavihek tudi svoj del v aplikaciji, kar pomeni svoj kontroler, pogled in model.



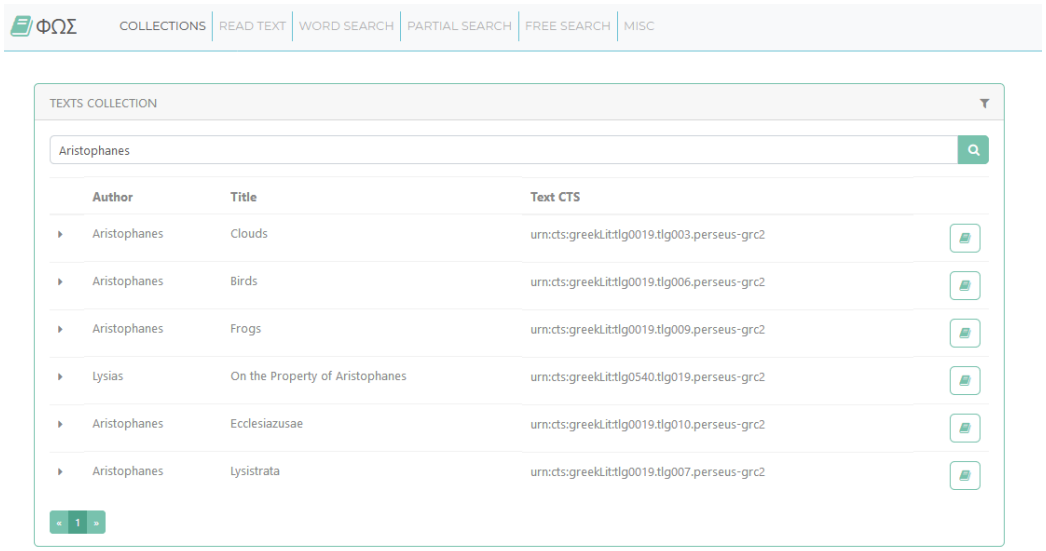
Slika 5.2: Glavna navigacija spletne aplikacije

5.2.1 Pregled zbirke tekstov

Najprej smo se lotili najbolj očitne oziroma najbolj preproste funkcionalnosti, to je prikaza seznama dokumentov iz indeksa. V ta namen smo v knjižnici za komunikacijo pripravili metodo, ki vrne deserializiran seznam dokumentov. Seznam zaradi hitrejšega delovanja vrne samo metapodatke posameznih dokumentov, torej avtorja, naslov in identifikator teksta.

Zgolj zaradi lepšega prikaza maske ob prihodu na naslov pregledovalnika zbirke uporabniku prikažemo naključnih pet tekstov iz korpusa. Nato lahko uporabnik vnese iskalni niz, na podlagi katerega mu vrnemo tekste, filtrirane po avtorju in naslovu. Na sliki 5.3 je primer iskanja po nizu „Aristophanes“. Zaradi možnega velikega števila zadetkov je bilo potrebno, tako kot na vseh ostalih maskah, kjer prikazujemo rezultate, urediti tudi paginacijo. V ta namen ima Elasticsearch v poizvedbah dodane temu namenjene parametre,

in sicer „od“ (angl. from) in „velikost“ (angl. size), s katerima lahko natančno določimo, kateri kos celotnega rezultata naj vrne klic.



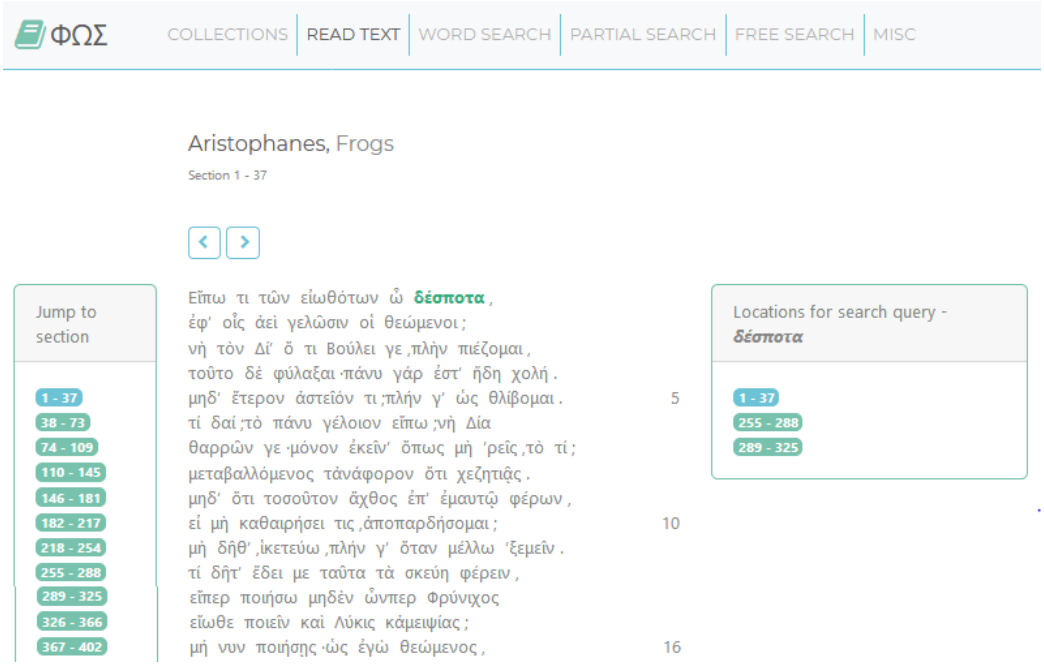
Slika 5.3: Rezultat iskalnega niza „Aristophanes“ na pregledovalniku zbirke tekstov

5.2.2 Bralnik tekstov

Naslednja na vrsti je bila implementacija maske, na kateri lahko uporabnik bere izbrani tekst. Na tej točki razvoja je bilo potrebno tekst iz dokumenta v Elasticsearchu uporabniku predstaviti v berljivi obliki. Tekst je v dokumentih namreč podan povsem razčlenjeno, tako da je vsaka beseda in vsako ločilo v ločenem vozlišču XML, kot je vidno na zadnjem primeru dokumenta na sliki virov 4.1. Na srečo so viri opremljenimi z zadostnimi podatki, da smo tekst lahko restavrirali v originalno obliko, kar je bilo potrebno, saj ima predvsem poezija vrstice in odstavke točno določene, tako da se lahko na njih sklicuje do vrstice natančno.

Vsaki besedi oziroma ločilu v viru je pripet atribut, ki nosi podatek, v katerem odstavku oziroma kitici se nahaja. Na podlagi tega atributa se tekst

najprej razdeli na sekcije, do katerih se lahko, kot je vidno na spodnji sliki 5.4, prosto dostopa na levem delu zaslona, oziroma se lahko pomakne na naslednjo ali prejšnjo sekcijo z gumboma naprej in nazaj. Nekatere sekcije so v vsebinskem smislu napačno označene, kar pa je posledica napak v samem viru podatkov. Napake bi bilo potrebno odpraviti v originalnih dokumentih XML pred indeksiranjem ali pa zgraditi uporabniški vmesnik, namenjen popravku že indeksiranih dokumentov. Obe možnosti bi zahtevali precej dodatnega dela in jih ni bilo mogoče realizirati v obsegu diplomskega dela.



Slika 5.4: Prikaz prvih kitic Aristofanovega dela Žabe

Ko so zgrajene sekcije, aplikacija sestavi še tekst iz trenutno izbrane sekcije v berljivo obliko. Ker je oblika besedila odvisna od vrste besedila, smo potrebovali podatek, ali je posamezen tekst proza ali poezija. Na tem koraku so pomagali raziskovalci iz oddelka za klasično filologijo na Filozofski fakulteti in nam pripravili seznam, kjer so za vse dokumente v korpusu definirali, ali gre za prozo ali za poezijo. Ta podatek smo potem pripeli Elasticsearchovim dokumentom. Razlika med prozo in poezijo je predvsem v tem, da smo

na bralniku teksta pri poeziji na koncu vsake pete vrstice dodali številko vrstice. Sam tekst se iz dokumentov sestavlja besedo po besedo v vrstice oziroma odstavke. Dolžino posamezne vrstice določimo iz vrednosti atributa „p“, ki ga nosi posamezna beseda, kot je vidno na sliki vira 4.2. Dokler je vrednost „p“ enaka, se vrstica nadaljuje. Pri poeziji so bile vrstice kratke oziroma je bilo število besed znotraj iste vrednosti „p“ malo, pri prozi pa je vrednost „p“ predstavljala odstavek, tako da se je vrstica na sami maski zaradi dolžine praviloma večkrat prelomila, kar pomeni, da na koncu dobimo lepo oblikovane odstavke glede na širino zaslona uporabnika.

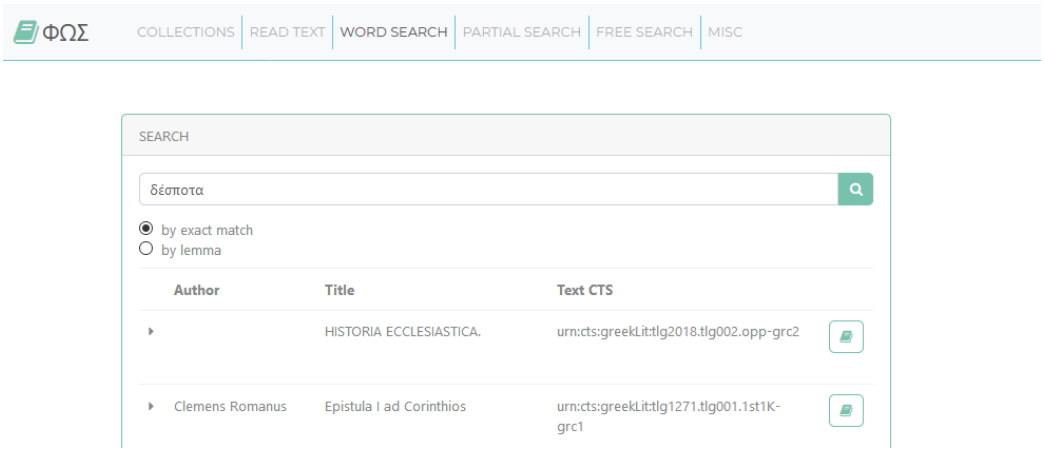
Bralnik je po prvih testiranjih uporabnikov dobil še funkcijo, da uporabniku dvoklik izbrane besede v novem oknu odpre slovar Termania z rezultatom iskanja izbrane besede. To uporabniku omogoča hitrejše delo oziroma učenje, saj besede ni potrebno prepisovati v slovar, prav tako pa je uporabno za ciljne uporabnike na Filozofski fakulteti v Ljubljani, saj je slovar slovenski. Ta funkcija se je skozi razvoj še nadgradila. Namesto tega, da se odpre slovar z izbrano besedo, se odpre z njeno lemo, kar je dodatna pridobitev za uporabnike, ker so v slovarju gesla prav tako vnesena v osnovnih oblikah in iskanje po neosnovnih oblikah ne vrača rezultatov.

Naknadno je bil implementiran tudi prikaz in navigacija po zadetkih. Na kasnejših zavihkih je namreč možno iskati dokumente po besedah ali delih besede, najdene zadetke v besedilu pa potem v aplikaciji prikažemo na desnem delu maske, kot je vidno na zgornji sliki 5.4. Zadetki na trenutni strani se obarvajo, hkrati pa je možno preskočiti na vsako od sekcij, kjer se najdena beseda ponovi.

5.2.3 Iskalniki

Ena od osnovnih zahtev je bilo iskanje po besedišču z določenim geslom. V ta namen je bilo razvitih več različnih iskalnikov. Najprej je bilo razvito najbolj osnovno iskanje, to je iskanje po celi besedi. Tako iskanje je bilo tudi najmanj zahtevno za sam Elasticsearch, saj dobro izkorišča členjenje v Elasticsearchu in izvaja preprosto ujemanje po celem nizu znakov. Uporabnik ima možnost

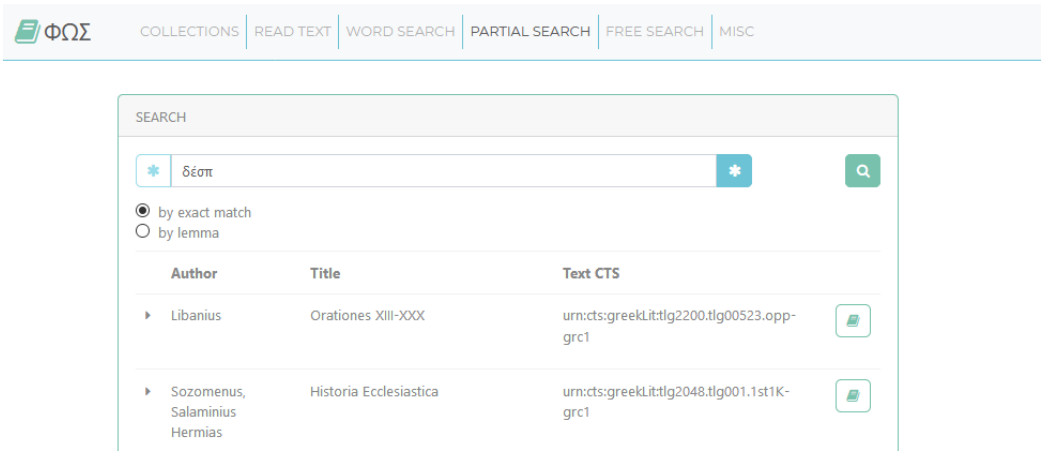
izbire in se lahko odloči, ali naj iskalnik išče po točni besedi, ali pa naj išče po lemah besed. Na sliki 5.5 smo na primer izbrali iskanje po točni obliki besede. Po izbiri dokumenta v seznamu zadetkov uporabnika preusmerimo na bralnik teksta, kjer se lahko s pomočjo menija na desni strani pomika po vseh sekcijah, kjer je iskalnik našel želeni niz. Tudi če je bilo izbrano iskanje po lemi, se uporabniku obarvajo tiste neosnovne oblike besede, katerih lema je bila najdena.



Slika 5.5: Iskalnik po celih besedah

Naslednji je bil implementiran iskalnik po delih besede. Ta se od iskalnika po celih besedah razlikuje v tem, da je možno izbrati poljuben konec ali začetek iskanega niza. Če želimo tudi besede, ki se začnejo s poljubnim nizom, označimo zvezdico na začetku vnosnega polja in podobno za besede, ki se poljubno končajo, zvezdico na koncu vnosnega polja. Na primeru na sliki 5.6 tako iščemo besede, ki se začnejo z vnesenim nizom znakov in se poljubno nadaljujejo. V bistvu je bila to implementacija iskanja z nadomestnim znakom, torej delnega ujemanja vnesenega niza. Tako kot na osnovnem iskalniku lahko uporabnik izbira med iskanjem po lemi oziroma po neosnovni obliki.

Kot zadnje je na vrsto prišlo tako imenovano „prosto iskanje“. Prosto iskanje najde vse take tekste, ki vsebujejo besede z znaki iz vnesenega niza,



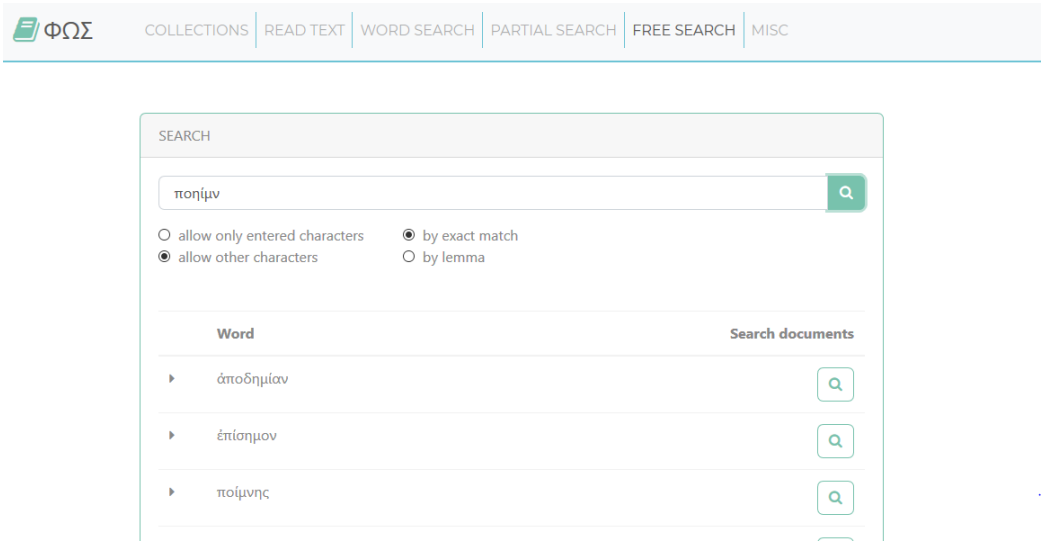
Slika 5.6: Iskalnik po delih besed

ne glede na vrstni red. Poleg tega se uporabnik lahko odloči, ali naj iskalnik najde samo take besede, ki vsebujejo izključno vnesene znake, ali pa tudi besede z znaki, ki jih ni v iskanem nizu. Tako na primer v prvem primeru z iskalnim nizom „abc“ najdemo besedi „bac“ in „cbbaa“, ne pa tudi besed „abcd“ ali „abcddk“. V drugem primeru pa najdemo vse naštete besede.

Pri tem primeru iskanja ni bilo možno uporabiti osnovnih poizvedb Elasticsearch, ki iščejo neposredno po besedah v dokumentih, tako kot smo to storili v prvih dveh iskalnikih, ampak si je bilo potrebno pomagati z Elasticsearchovimi agregacijami. Agregacije nam omogočajo, da posamezna polja v indeksu združimo pod nekimi pogoji.

Če torej združimo vse besede iz vseh dokumentov v indeksu tekstov brez pogojev, dobimo seznam z eno pojavitvijo vsake možne besede v indeksu. Iz dobljenega seznama potem dobimo besede, ki ustrezajo iskanju. Pri tem si pomagamo z množicami. Če gre za prvo, ekskluzivno iskanje, naredimo presek besede in iskalnega niza. Če je število znakov v preseku enako številu različnih znakov v besedi in številu različnih znakov v iskanem nizu, potem se beseda ujema, sicer ne. Če pa gre za drugo iskanje, potem se mora število znakov v narejenem preseku ujemati samo s številom znakov v iskanem nizu. Najdene besede potem prikažemo uporabniku in to je tudi celotni prvi korak

prostega iskanja, kot je vidno na sliki 5.7.



Slika 5.7: Prvi korak prostega iskanja

Po tem si uporabnik izbere eno od najdenih besed in aplikacija izvede iskanje po dokumentih za to besedo. Še pred tem lahko uporabnik izbere, ali naj se iskanje zgodi po točnih besedah ali pa po lemah. Najdene rezultate potem prikažemo uporabniku, ta pa lahko, enako kot na ostalih iskalnikih, izbere dokument, kar ga preusmeri na branje izbranega teksta s prikazom zadetkov iskane besede.

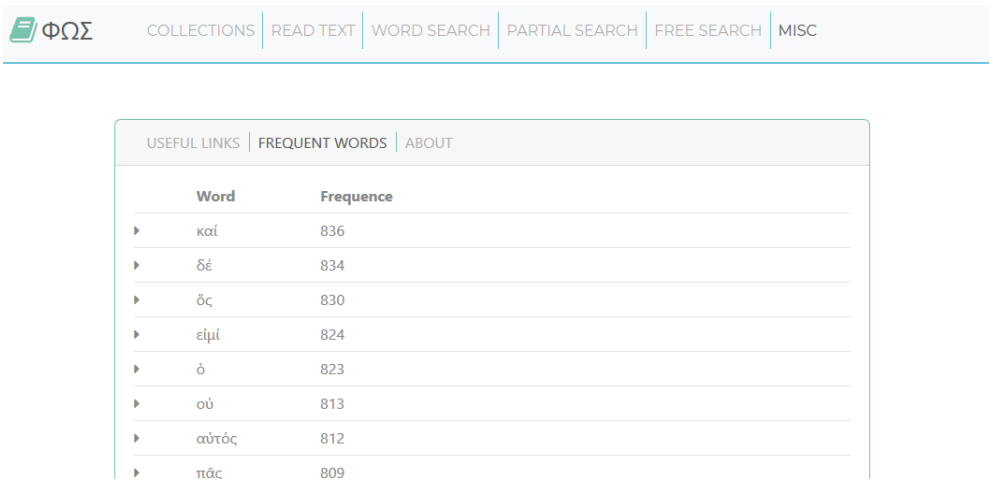
5.2.4 Pogoste besede in ostalo

Na zadnjem zavihku v aplikaciji najdemo implementacijo še zadnjih dveh želja uporabnikov. Prva je preprost seznam povezav do drugih aplikacij za pomoč pri učenju stare grščine. V nastavitveni datoteki Web.config za aplikacijo MVC je možno v seznam dodajati nove povezave, brez da bi bilo potrebno aplikacijo še enkrat prevesti.

Druga želja pa je bila, da je v aplikaciji možno dostopati do seznama najpogostejših lem celotnega korpusa. V ta namen smo postavili nov in-

deks v Elasticsearchu, v katerega smo v končni fazi shranili seznam tisoč najpogostejših lem v korpusu. Seznam, katerega začetek je viden na sliki 5.8, smo sestavili podobno kot pri prostem iskanju, torej z agregacijo nad lemami. Najprej pridobimo agregacijo vseh lem, jo sortiramo po pogostosti in prvih tisoč besed indeksiramo. Nov indeks s tem seznamom je bil vzpostavljen predvsem zaradi hitrosti dostopa do seznama najpogostejših lem, saj je generiranje le-tega ob vsakem zahtevku relativno zahtevna in počasna operacija.

V primeru da se v indeks dokumentov dodaja nove tekste, je seveda potrebno seznam najpogostejših lem osvežiti. V ta namen smo pripravili regeneriranje indeksa najpogostejših lem iz spletne aplikacije. Zahtevek za regeneriranje se tako lahko sproži s klikom na gumb v administratorskem delu aplikacije (spletnemu naslovu aplikacije dodamo /Admin). Po kliku najprej izbrišemo vse vnose iz indeksa in potem, enako kot prvič, pripravimo agregacijo in indeksiramo najpogostejše pojavitve lem.



Slika 5.8: Zavihek „Misc“ in seznam najpogostejših lem v besedišču

Poglavje 6

Sklepne ugotovitve

V sklopu diplomskega dela smo najprej na spletu raziskali možne vire za aplikacijo, izbrani vir predelali in ga shranili v iskalnem strežniku Elasticsearch. Potem smo razvili spletno aplikacijo za pomoč pri učenju in poučevanju stare grščine, ki izbrane vire prikazuje v berljivi obliki in omogoča iskanje po besedilih. Spletni del aplikacije je bil razvit v ogrodju .NET 4.5 in MVC5, za razvoj pa smo primarno uporabljali Visual Studio 2015. Želeli smo, da ima aplikacija minimalističen in sodoben videz, za kar je poskrbel predelan stil CSS Bootstrap. V diplomskem delu predstavimo tudi ostale podobne aplikacije na spletu in primerjamo obstoječe aplikacije z našo končno verzijo.

Funkcionalnosti aplikacije so bile znane že pred začetkom razvoja. Pripravljene so bile v sodelovanju z raziskovalci z oddelka za klasično filologijo na Filozofski fakulteti Univerze v Ljubljani. Kasneje smo funkcionalnosti na podlagi povratnih informacij s testiranja uporabnikov še dograjevali in izpopolnjevali, tako da smo prišli do trenutne končne verzije.

Seveda je kot pri vsaki aplikaciji možnosti za nadgradnjo še precej. Dodana vrednost bi bila na primer pri izbiri besede v bralniku primerjava z novo grščino. Prav tako bi lahko prikaz rezultata iz slovarja prikazali kar v uporabniškem vmesniku bralnika teksta. Ideja je bila tudi, da se v aplikacijo doda igre, ki bi spodbujale študente k pomnjenju besedišča in nepravilnih oblik. Kar se tiče uporabniške izkušnje, bi lahko na primer združili vse

tri različne iskalnike v enem uporabniškem vmesniku, namesto paginacije bi lahko v prikazovalnikih rezultatov uporabili neskončni drsnik in podobno. Nadgradnje aplikacije smo omogočili s tem, da je aplikacija odprtokodna in prosto dostopna na repozitoriju Git.

Literatura

[1] Amebis d. o. o. Kamnik. Slovar Termania, spletni slovarski portal. Dosegljivo: https://www.termania.net/. [Dostopano: 30. 11. 2018].

[2] Project Gutenberg Literary Archive Foundation. Projekt gutenberg, digitalna knjižnica. Dosegljivo: https://www.gutenberg.org/. [Dostopano: 30. 11. 2018].

[3] Gregory R. Crane (ured.). Perseus, digitalna knjižnica. Dosegljivo: http://www.perseus.tufts.edu/hopper/. [Dostopano: 30. 11. 2018].

[4] Maria Pantelia (ured.). Thesaurus linguae graecae®, digitalna knjižnica grške literature. Dosegljivo: http://stephanus.tlg.uci.edu/. [Dostopano: 30. 11. 2018].

[5] Shay Banon (ustanovitelj). Elasticsearch. Dosegljivo: https://www.elastic.co/products/elasticsearch. [Dostopano: 30. 11. 2018].

[6] Apache Software Foundation. Lucene. Dosegljivo: http://lucene.apache.org/. [Dostopano: 30. 11. 2018].

[7] Db-engine, trend popularnosti iskalnih strežnikov. Dosegljivo: https://db-engines.com/en/ranking_trend/search+engine. [Dostopano: 30. 11. 2018].

[8] Microsoft. Ogrodje .net. Dosegljivo: https://en.wikipedia.org/wiki/.NET_Framework. [Dostopano: 30. 11. 2018].

[9] Microsoft. Asp.net mvc 5. Dosegljivo: https://docs.microsoft.com/en-us/aspnet/mvc/mvc5. [Dostopano: 30. 11. 2018].

[10] Elastic. Elasticsearch.net, nizkonivojski elasticsearch odjemalec. Dosegljivo: https://github.com/elastic/elasticsearch-net-example/. [Dostopano: 30. 11. 2018].

[11] Elastic. Nest, visokonivojski elasticsearch odjemalec. Dosegljivo: https://github.com/elastic/elasticsearch-net/tree/master/src/Nest. [Dostopano: 30. 11. 2018].

[12] Microsoft. Intellisense. Dosegljivo: https://docs.microsoft.com/en-us/visualstudio/ide/using-intellisense?view=vs-2017. [Dostopano: 30. 11. 2018].

[13] Progress Software Corporation. Fiddler, http razhroščevalnik. Dosegljivo: https://www.telerik.com/fiddler. [Dostopano: 30. 11. 2018].

[14] Man-in-the-middle attack. Dosegljivo: https://en.wikipedia.org/wiki/Man-in-the-middle_attack. [Dostopano: 30. 11. 2018].

[15] Leipzig University. First1kgreek, projekt digitalizacije starogrških besedil. Dosegljivo: http://opengreekandlatin.github.io/First1KGreek/. [Dostopano: 30. 11. 2018].

[16] Giuseppe Giovanni Antonio Celano. Tokenized and sentence-splitted ancient greek texts, repozitorij digitaliziranih starogrških tekstov. Dosegljivo: https://github.com/gcelano/CTSAncientGreekXML. [Dostopano: 30. 11. 2018].

[17] Giuseppe Giovanni Antonio Celano. Lemmatized ancient greek texts, repozitorij digitaliziranih starogrških tekstov. Dosegljivo: https://github.com/gcelano/LemmatizedAncientGreekXML. [Dostopano: 30. 11. 2018].