# Razvoj RESTFUL web usluge i aplikacijskog programskog sučelja u .NET okviru

**Marić, Petar**





**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU**
**FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA**

**Sveučilišni studij**

# RAZVOJ RESTFUL WEB USLUGE I APLIKACIJSKOG PROGRAMSKOG SUČELJA U .NET OKVIRU

**Završni rad**

**Petar Marić**

**Osijek, 2018.**

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

**Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju**

**Osijek, 04.09.2018.**

**Odboru za završne i diplomske ispite**

# Prijedlog ocjene završnog rada

| | |
|---|---|
| **Ime i prezime studenta:** | Petar Marić |
| **Studij, smjer:** | Preddiplomski sveučilišni studij Računarstvo |
| **Mat. br. studenta, godina upisa:** | R3801, 19.09.2017. |
| **OIB studenta:** | 33866030341 |
| **Mentor:** | Prof.dr.sc. Goran Martinović |
| **Sumentor:** | Dr.sc. Bruno Zorić |
| **Sumentor iz tvrtke:** | |
| **Naslov završnog rada:** | Razvoj RESTFUL web usluge i aplikacijskog programskog sučelja u .NET okviru |
| **Znanstvena grana rada:** | **Programsko inženjerstvo (zn. polje računarstvo)** |
| **Predložena ocjena završnog rada:** | Izvrstan (5) |
| **Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:** | Primjena znanja stečenih na fakultetu: 3 bod/boda<br>Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda<br>Jasnoća pismenog izražavanja: 3 bod/boda<br>Razina samostalnosti: 3 razina |
| **Datum prijedloga ocjene mentora:** | 04.09.2018. |
| **Datum potvrde ocjene Odbora:** | 12.09.2018. |

| | |
|---|---|
| Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija: | Potpis: |
| | Datum: |

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

# IZJAVA O ORIGINALNOSTI RADA

**Osijek, 16.09.2018.**

| | |
|---|---|
| **Ime i prezime studenta:** | Petar Marić |
| **Studij**: | Preddiplomski sveučilišni studij Računarstvo |
| **Mat. br. studenta, godina upisa:** | R3801, 19.09.2017. |
| **Ephorus podudaranje [%]:** | 7 |

Ovom izjavom izjavljujem da je rad pod nazivom**: Razvoj RESTFUL web usluge i aplikacijskog programskog sučelja u .NET okviru**

izrađen pod vodstvom mentora Prof.dr.sc. Goran Martinović

i sumentora Dr.sc. Bruno Zorić

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

# SADRŽAJ

# 1. UVOD

Internet danas ima veliku ulogu u životu ljudi na cijeloj planeti što govori podatak da je broj korisnika na početku 2018. godine nadmašio brojku od četiri milijarde [1]. Svoje začetke vidio je u znanstvenim laboratorijima i sveučilištima u Sjedinjenim Američkim Državama tijekom 60-ih godina prošloga stoljeća tada zvan ARPANET. Prva uspješna komunikacija ostvarena je 1969. godine između Stanford i UCLA sveučilišta u Kaliforniji te od tog trenutka počinje eksponencijalno geografsko i funkcionalno širenje Interneta. Osnivaju se različita nadležna tijela koja donose standarde i protokole vezane za fizički, mrežni, transportni i aplikacijski sloj Interneta te osiguravaju njihovo provođenje. Neka od najvažnijih takvih tijela su IANA, IETF, ICANN, W3C. Internet je u današnje vrijeme postao sinonim za tehnologiju i tehnološki napredak. Prije samo stotinu godina vrijeme potrebno da se ostvari potpuna obostrana razmjena poruka između pošiljatelja u Europi i primatelja u Sjevernoj Americi bilo je oko dva mjeseca. Vraćanjem u sadašnjost, vrijeme potrebno za obostranu razmjenu poruka između bilo koje dvije točke na planeti Zemlji ili čak između satelita u geostacionarnoj orbiti i površine Zemlje je manje od jedne sekunde. Internet zapravo predstavlja skup međusobno povezanih osobnih računala, prijenosnih računala, mobilnih uređaja, *mainframe* računala, poslužitelja i ostalih uređaja koji su povezani pomoću bakrenih žica, koaksijalnih kabela, optičkih kabela te bežične tehnologije u osobne, poslovne, edukacijske, medicinske ili vojne svrhe. Servisi koje podržava Internet su globalna mreža za razmjenu informacija (engl. *World Wide Web*; WWW), elektronička pošta (engl. *e-mail*), *telnet*, mrežne novine, protokol za razmjenu datoteka (engl. *File Transfer Protocol*; FTP) te komunikacija glasom putem internet protokola (engl. *Voice over Internet Protocol*; VoIP). Na globalnoj mreži za razmjenu informacija također postoji mnoštvo servisa. Servisi predstavljaju kod koji se pokreće na nekom udaljenom računalu (poslužitelju). Zatim tim servisima mogu pristupati ljudi, računala ili drugi servisi (klijenti) s ciljem dobivanja povratnih informacija ili izvršavanja određenih zadataka.

Tema ovoga rada bit će izlaganje elemenata i stilova koji su osnova REST arhitekture te RESTful web servisa koji danas svojom uporabom prevladavaju zbog jednostavnosti korištenja i kreiranja. U drugom poglavlju ovog rada opisani su osnovni pojmi koji omogućuju postojanje web servisa, dan je pregled elemenata potrebnih za kreiranje SOAP web servisa te je opisana povijest REST arhitekture, elementi i stilovi iz kojih je nastala te ograničenja nužna za postojanje RESTful servisa. Također će se u drugom poglavlju dati pregled dvaju bitnih formata koji se koriste za generiranje zahtjeva i odgovora poslužitelja na zahtjeve, tj. o JSON (engl. *JavaScript Object*

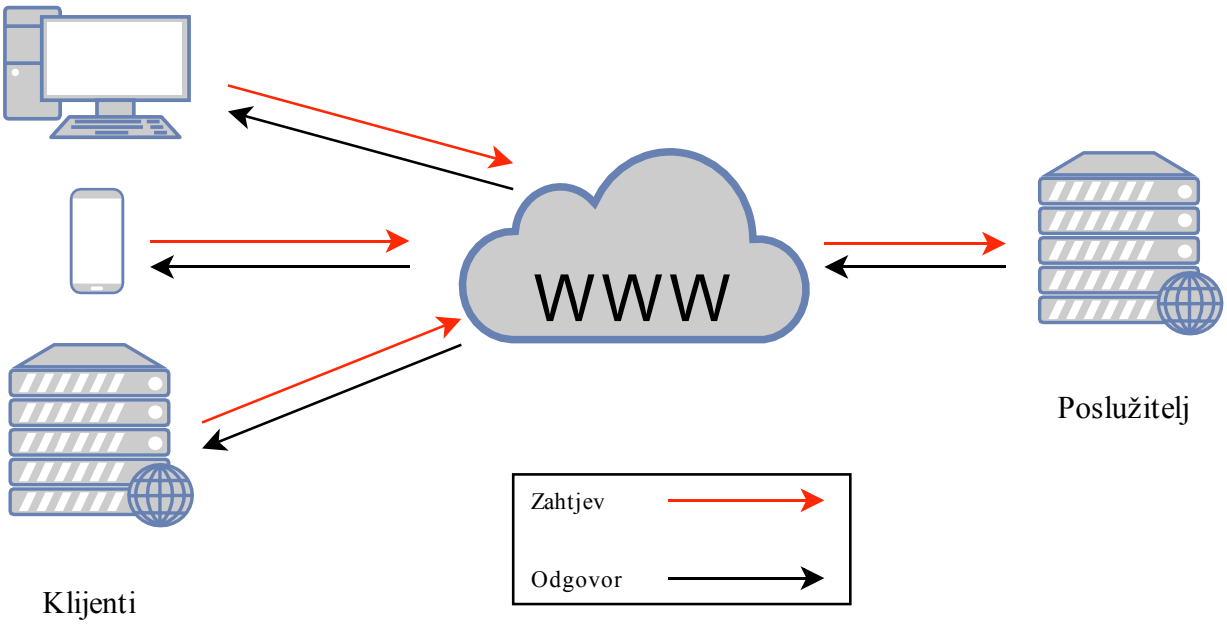
*Notation*) formatu te o formatu proširivog označnog jezika (engl. *Extensible Markup Language; XML*). U trećem poglavlju ostvareno je i opisano programsko rješenje za pohranu i pretraživanje sadržaja vijesti unutar Elasticseracha uz kreiranje vlastitog aplikacijskog programskog sučelja (engl. *Application Programming Interface*; API) razvijenog unutar .NET okruženja. U trećem poglavlju također su opisani korišteni alati i tehnologije prilikom razvoja programskog rješenja. Konačno poglavlje donosi zaključke izvedene iz potpoglavlja i odlomaka kojima će biti izloženi najbitniji pojmovi vezani za SOAP i REST web servise te ostvarenje praktičnog dijela rada i radovi koji kao osnovu mogu uzeti ovaj rad.

## 1.1. Zadatak završnog rada

Web usluge postale su uobičajeno rješenje za razmjenu znanja i informacija među heterogenim sustavima. U teorijskom dijelu rada potrebno je opisati različite tehnologije i protokole koji omogućuju web usluge. Naglasak staviti na REST arhitekturu, njene prednosti i nedostatke. U praktičnom dijelu rada ostvariti programsko rješenje za pretragu sadržaja pohranjenog u bazi podataka korištenjem .NET okvira.

# 2. WEB USLUGE

Web servis predstavlja računalni kod koji se pokreće na udaljenom računalu. Web servis treba biti formatiran tako da pruža svoje funkcionalnosti preko vlastitog definiranog sučelja koje koristi određene podatkovne formate i standarde za komunikaciju (izmjenu poruka). Web servisima je moguće pristupiti izvana, tj. s Interneta i intraneta koji može biti svojstven za određenu organizaciju ili akademsku ustanovu. Najčešće korišteni oblik web servisa jest klijent-poslužitelj koji je prikazan slikom 2.1.




Slika 2.1. *Prikaz principa rada web servisa*

## 2.1. Osnovni pojmovi

Svaki web servis ima određeno sučelje preko kojega komunicira s okolinom, a dobrim oblikovanjem sučelja omogućava se lakši razvoj funkcionalnosti servisa, izmjena postojeće funkcionalnosti bez izmjene sučelja te povećanje broja pruženih usluga. Sučelje omogućava servisu neovisnost o odabiru programskog jezika koji definira njegovu funkcionalnost te također omogućava klijentima neovisnost uporabe programskih jezika za definiranje vlastitih aplikacija. U današnje vrijeme postoje dvije najkorištenije arhitekture za razvoj web servisa, a to su SOAP i REST koje će u nastavku ovog rada biti detaljnije opisane. Oba tipa arhitekture za definiranje prijenosa informacija koriste HTTP protokol i URI standard za identifikaciju servisa u slučaju SOAP ili resursa u slučaju REST arhitekture.

### 2.1.1. HTTP protokol

HTTP (engl. *Hypertext Transfer Protocol*) predstavlja temeljni protokol unutar globalne mreže za razmjenu informacija koji omogućava prijenos hipertekstualnih datoteka definiranih HTML označnim jezikom ili ostalim podržanim tipovima podataka. Prema [2], originalno ga je osmislio Tim Berners Lee dok je radio u CERN-u 1990. godine. HTTP prema TCP/IP Internet modelu pripada aplikacijskom sloju. U svrhu potpore web servisa koristi strogo formatirane poruke. Svaka poruka sastoji se od elementa zaglavlja (engl. *Header*) i elementa tijela (engl. *Body*). U elementu zaglavlja u prvom retku mora biti definirana HTTP metoda koja se upućuje poslužitelju, putanja URI-ja do traženog resursa te inačica HTTP protokola koju klijent koristi. U elementu zaglavlja također se nalazi specificiran autoritet iz kojeg će se zajedno s relativnom putanjom moći odrediti IP adresa poslužitelja pomoću DNS sustava (engl. *Domain Name System*) te proslijediti putanju do resursa. Ostali elementi zaglavlja koji imaju bitnu ulogu pri korištenju HTTP-a za komunikaciju između web servisa su: tip podatka koji je sadržan unutar elementa tijelo (*Content-Type*), upravljanje priručnom memorijom bilo na poslužitelju, klijentu ili posredniku (*Cache-Control*), datum i vrijeme kreiranje zahtjeva/odgovora (*Date*), zadnje vrijeme modifikacije resursa (*Last-Modified*) i mnogi drugi. U slučaju stavljanja sadržaja u zahtjev ili stavljanje sadržaja obrade web servisa u odgovor potrebno je specificirati tip sadržaja u zaglavlju te staviti taj sadržaj u element tijela HTTP poruke koji je od zaglavlja razdvojen jednim praznim retkom. HTTP također podržava skup standardnih metoda za dohvaćanje sadržaja (GET), kreiranje novog sadržaja ili formatiranje složenijeg upita na web servis (POST), izmjenu postojećeg resursa (PUT) te brisanje elemenata (DELETE). Za informiranje klijenta na određene zahtjeve koristi definirane statusne poruke koje su u rasponu od 1XX do 5XX.

### 2.1.2. Jedinstveni identifikator resursa

Standard za jedinstveni identifikator resursa (engl. *Uniform Resource Identifier;* URI) koristi se za jednoznačno definiranje resursa unutar cijele globalne mreže za razmjenu informacija. Prema [3], sastoji se od elemenata sheme koji specificiraju protokol korišten za pristup resursu, autoriteta tj. poslužitelja kojem se želi pristupiti, hijerarhijske putanje koja vodi do resursa te od dijela upita koji započinje znakom upitnika. Upit sadržava parove *ključ-vrijednost* (engl. *key-value pair*) koje poslužitelj interpretira drugačije od hijerarhijske putanje do resursa, a imaju svrhu pružiti dodatne parametre koji će se koristiti pri radu web servisa tj. generiranju odgovora. Upit je od putanje odvojen znakom upitnika '?'. Na slici 2.2. označeni su navedeni dijelovi zamišljenog URI-ja nekog RESTful servisa. Takvim URI-jem bi se omogućila pretraga filmova pod hijerarhijskom

strukturom *filmovi*. S ciljem pojašnjenja koncepta, odabran je film *Inception* te se servisu naznačuje da taj film započne na 512. sekundi te da pauzira reprodukciju filma.


Slika 2.2. *Prikaz komponenti URI-ja*

Prema [3], URI predstavlja uniju URN (engl. *Uniform Resource Name*) i URL (engl. *Uniform Resource Locator*) standarda za imenovanje i lociranje resursa na globalnoj mreži za razmjenu informacija. Iz toga slijedi da URI prikazan na slici 2.2. odgovara i URL-u kojim se može pristupiti traženom resursu.

## 2.2. SOAP arhitektura

SOAP arhitektura web servisa sastoji se od tri ključna dijela: jezika za opisivanje web servisa koji predstavlja standard za opisivanje mogućnosti web servisa, UDDI standarda za objavljivanje servisa te SOAP protokola za komunikaciju između web servisa.

### 2.2.1. Jezik za opisivanje web servisa

Prema [4], jezik za opisivanje web servisa (engl. *Web Services Description Language*; WSDL) predstavlja standard za kreiranje dokumenata kojima se opisuju servisi pomoću XML formata. Takav dokument u sebi sadrži opis svih metoda koje sadrži određeni web servis i odgovora koji se mogu očekivati od svake završne točke servisa. Prema [5], mora sadržavati oznake tipova podataka korištenih u servisu (engl. *types*), oznaku poruke u kojoj se opisuju podaci koji se prenose (engl. *messages*), oznaku vrata pomoću koje opisuju operacije nad podacima i poruke koje obuhvaćaju (engl. *port type*) te oznaku vezivanja kojom se označava korišteni protokol i njegove detalje (engl. *binding*). WSDL je većinom namijenjen računalima za automatsko generiranje programskog koda iz WSDL dokumenta.

### 2.2.2. UDDI standard

Prema [4], UDDI (engl. *Universal Description, Discovery, and Integration*) predstavlja standard za publiciranje i otkrivanje vlastitog servisa široj javnosti. Predstavlja repozitorij, odnosno registar u kojega se mogu upisivati zapisi postojećih web servisa opisanih WSDL-om, a svaki pružatelj web servisa može odlučiti ne sudjelovati u javnoj objavi svojega web servisa. Klijenti mogu tražiti pogodne servise u navedenom repozitoriju te kada nađu odgovarajući servis, mogu preuzeti WSDL

dokument u kojem je opisana cjelokupna funkcionalnost web servisa. UDDI predstavlja sustav koji ima tri kategorije u koje klasificira servise:

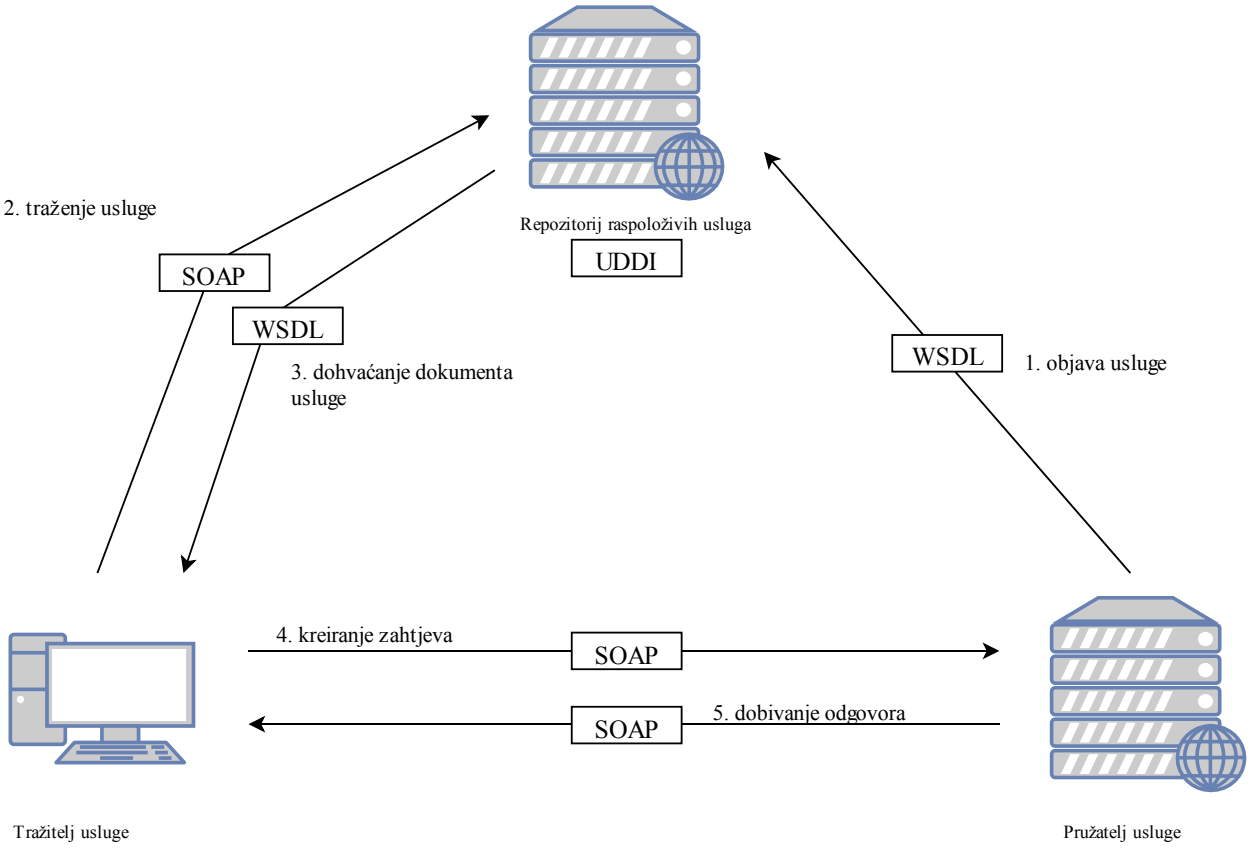
1. Bijele stranice: Sastoji se od organizacija i servisa koje pružaju.
2. Žute stranice: Klasificira servise u određene grupe funkcionalnosti koju obavljaju.
3. Zelene stranice: Sadrže poveznice na WSDL dokumente kako bi se servisi mogli koristiti.

### 2.2.3. SOAP protokol

SOAP je skraćenica od *Simple Object Access Protocol*. Prema [6], distribuirani servisi su ranije koristili CORBA (engl. *Common Object Request Broker Architecture*) i DCOM (engl. *Distributed Component Object Model*) tehnologije bazirane na ORPC (engl. *Object Remote Procedure Call*) protokolima koji su htjeli povezati objektnu orijentiranost programskih jezika i mrežne protokole. ORPC je zapravo predstavljao identifikator koji poslužitelj koristi za odabir traženog objekta. Slabosti takvog pristupa su u tome što se na strani klijenta i na strani poslužitelja mora implementirati isti distribuirani objektni model (Java/RMI ili CORBA/IIOP) te što su vatrozidi i posrednici često konfigurirani da blokiraju IIOP i RMI protokole. Kao odgovor na DCOM i CORBU pojavljuje se SOAP koji služi za razmjenu poruka između heterogenih računalnih sustava. SOAP poruka prenosi se putem protokola aplikacijskog sloja – HTTP ili SMTP protokola, no preferira se HTTP zbog svoje utjecajne prisutnosti u globalnoj mreži za razmjenu informacija i konfiguriranih vatrozida koji uobičajeno propuštaju promet HTTP protokola. SOAP protokol definira format poruke koji se koristi za komunikaciju između klijenta i servisa te skup pravila kojem će se iz SOAP poruke prepoznati i pozvati metoda te kako će se rezultat obrade te metode vratiti u SOAP poruku prema klijentu. Svaka SOAP poruka definirana je omotnicom (engl. *envelope*) koja je definirana XML formatom opisanim u potpoglavlju 2.4. Obavezni dio omotnice predstavlja tijelo omotnice (engl. *body*) koje sadržava informacije potrebne za izvršavanje željenog zadatka. Omotnica također može sadržavati element zaglavlja u kojem se definiraju oznake vezane za obradu informacija prolaskom kroz posrednike između tražitelja usluge i pružatelja tražene usluge. Tijelo omotnice može sadržavati informacije vezane za poziv metode te njezine parametre ili poruke o greškama. Poruke o greškama mogu se pojaviti prilikom nepravilno formatiranog poziva metode ili grešaka koje mogu nastati na posrednicima ako ne prepoznaju parametre koji su u zahtjevu označeni kao obavezni. SOAP omotnica uobičajeno se stavlja unutar tijela HTTP protokola koji je odvojen od svojega zaglavlja jednim praznim retkom. To znači da se SOAP protokol koristi na sloju iznad aplikacijskog sloja u slojevitom sustavu mrežnih protokola. POST metoda HTTP protokola koristi se za slanje SOAP omotnica te primanje rezultata u omotnici vraćenih od servisa. PUT i DELETE HTTP metode nisu podržane specifikacijom [7]. SOAP

komponente također mogu prepoznati HTTP statusne kodove koje koriste kao dodatni mehanizam za sporazumijevanje [37].

Prema [8], prednosti SOAP arhitekture nad REST arhitekturom su povećana sigurnost uporabom *Web Service-Security* (*WS-Security*). *WS-Security* predstavlja proširenje SOAP arhitekture koje omogućava SOAP porukama povjerljivost, integritet, autentifikaciju korisnika, zaštitu od virusa itd. Iznimnu primjenu proširenje nalazi u poslovnoj i administrativnoj okolini. SOAP arhitektura također podržava ACID princip što za cilj ima očuvanje integriteta baza podataka prilikom rada s transakcijama. Nedostaci SOAP arhitekture su potreba za više mrežnog prometa od REST arhitekture te više potrebnog vremena za obradu zahtjeva i generiranje odgovora prema [9], nemogućnost spremanja SOAP odgovora u priručnu memoriju te izuzetno složen razvoj SOAP arhitekture ako se nalaže potreba korištenja raznih proširenja (*WS-**) SOAP arhitekture. Na slici 2.3. prikazane su radnje koje se moraju napraviti za omogućavanje korištenja određene usluge uporabom SOAP arhitekture. Sadržaj slike prilagođen je pomoću [39].




Slika 2.3. *Prikaz SOAP arhitekture*

## 2.3. REST arhitektura

Skraćenica REST dolazi od engleskog naziva *Representational State Transfer* što znači prijenos reprezentacijskog stanja. Roy Fielding je u svojoj disertaciji 2000. godine opisao novi arhitekturalni stil koji je predstavljao hibrid postojećih stilova i elemenata uz dodatna ograničenja koja definiraju takav stil. Stilovi unutar mrežnih arhitektura predstavljaju skup arhitekturalnih ograničenja koja nameću dozvoljene odnose između elemenata arhitekture i njihove značajke [11]. Također prema [11] arhitekturu objedinjuju tri tipa elemenata kojima se postižu željena arhitekturalna svojstva, a riječ je o komponentama, elementima konektorima i podataka. Komponente predstavljaju sustavi koji vrše manipulaciju elementima podataka, a komponente su međusobno povezane elementima konektora.

### 2.3.1. Povijest REST arhitekture

Prema [10], REST je rezultat suradnje Roya Fieldinga i Henrika Nielsena na razvoju specifikacije HTTP protokola inačice 1.1. Fielding je pri izradi specifikacije izradio HTTP *objektni model* pomoću kojega bi testirao utjecaj promjene protokola na primjenu unutar globalne mreže za razmjenu informacija. Prema [38], Fielding je svoj model nazvao *objektni model* pošto su u to vrijeme bili popularni razni objektni modeli. Namjera razvijenog *objektnog modela* bila je prikazati kako bi se aplikacije trebale ponašati na globalnoj mreži za razmjenu informacija. Fielding se 1995. godine pridružio razvojnom timu projekta *Apache HTTP Server* u kojemu je razvijena primjena HTTP protokola. Unutar jedne godine taj program postao je najpopularniji za programsku podršku poslužitelja. Projekt je 1999. godine dobio nagradu za korištenu arhitekturu i zbog toga što je razvijen na platformi otvorenog koda. Fielding je tek nakon izlaska HTTP/1.1 protokola 1997. godine odlučio u svojoj disertaciji istražiti i opisati HTTP objektni model na kojem je prethodno radio. Tek nakon konzultacije sa svojim kolegama mu je postalo jasno da je zapravo riječ o arhitekturalnom stilu koji zahvaća više različitih stilova mrežnih arhitektura. Nakon toga odlučio je promijeniti ime objektnog modela u *Representational State Transfer* te njegovo opisivanje. U nastavku slijedi opis elemenata REST arhitekturalnog stila, stilova mrežnih arhitektura iz kojih je deriviran REST arhitekturalni stil te ograničenja kojih bi se trebalo pridržavati ako se kao rezultat želi dobiti RESTful web servis.

### 2.3.2. Elementi

U ovome odjeljku bit će izloženi elementi koji čine REST arhitekturalni stil tj. elementi podataka, elementi konektora te komponente koje omogućavaju razmjenu informacija između distribuiranih hipermedijskih sustava.

**Elementi podataka**

Prema [11], u sustavima distribuiranih objekata podatke skrivaju komponente koje vrše nekakvu vrstu obrade nad tim podacima dok je u REST-u bitno stanje tih podataka. U distribuiranom sustavu potrebno je prenositi podatke od mjesta gdje su pohranjeni do mjesta gdje se zatraži njihova prisutnost. Navedeno se ostvaruje poveznicom te su dizajneru takvog sustava omogućene tri opcije:

1. Može imati skrivenu implementaciju programa koji vrši obradu podataka na zahtjev te nakon prolaska podataka kroz takav program rezultat šalje kao odgovor na prethodno primljeni zahtjev.
2. Može u odgovor upakirati podatke i program potreban za prikazivanje tih podataka.
3. U odgovor može staviti neobrađene podatke i metapodatke koji opisuju tip neobrađenih podataka te se time omogućava klijentu odabir programa kojim će vršiti obradu nad podacima.

Prvom opcijom omogućava se skrivanje implementacije podataka, ali se ograničava mogućnost poslužitelja za posluživanjem većeg broja zahtjeva zbog potrebe vršenja lokalne obrade podataka na svaki zahtjev. Drugom opcijom potreban je prijenos većeg broja datoteka zbog toga što je potrebno upakirati i podatke i program koji razumije te podatke, no još uvijek omogućava skrivanje podataka. Trećom opcijom omogućava se poslužitelju posluživanje većeg broja zahtjeva zbog toga što ne mora vršiti nikakvu obradu nad podacima pošto ih šalje u običnom obliku te poslužitelj i klijent moraju moći razumjeti isti tip podatka kako bi mogli komunicirati. Jedini nedostatak takvog sustava jest nemogućnost skrivanja informacija.

REST arhitektura predstavlja hibrid tih triju opcija pošto svima omogućava razumijevanje tipova podataka, no ograničava što će biti vidljivo pomoću standardiziranog sučelja preko kojega bi se vršila komunikacija. Komponente unutar REST arhitekture mogu proizvoljno birati reprezentaciju resursa ovisno o standardnom tipu podatka ili prirodi resursa (npr. ako je kreirano sučelje za rad s audio datotekama birat će se FLAC, MP3, OGG reprezentacije, a ne JPEG ili PNG). Iza sučelja treba biti skriven originalni tip resursa koji se poslužuje kao reprezentacija ili iz kojega se deriviraju ostali topovi reprezentacija. Iz navedenog slijedi da REST odvaja odgovornost između

klijenta i poslužitelja te je omogućeno da se te dvije komponente razvijaju samostalno, da je moguće sakriti informacije iza sučelja te da se omogućava bolje korisničko iskustvo mogućim preuzimanjem dodatnih sadržaja s poslužitelja. U tablici 2.1. koja je preuzeta iz [11] prikazani su elementi podataka REST-a.

Tablica 2.1. *REST elementi podataka*

| Element podatka | Primjeri |
|---|---|
| Resurs | Namjeravana konceptualna meta hipertekst reference |
| Identifikator resursa | URI |
| Reprezentacija | XML, JSON, YAML, JPEG |
| Metapodaci reprezentacije | Tip medija (metoda zaglavlja HTTP protokola – *Content-Type*), zadnje vrijeme modifikacije, datum slanja kao odgovora na zahtjev |
| Metapodaci resursa | Poveznica izvora, alternative resursa |
| Kontrolni podaci | HTTP ugrađene metode za kontroliranje događaja poslužitelja: ako-odgovara (engl. *If-match*), kontrola priručne memorije (engl. *Cache-Control*) |

1. Resurs i identifikator resursa

Resurs u REST-u zapravo predstavlja apstrakciju koja se može odnositi na bilo što što se može imenovati, bilo materijalno ili nematerijalno. Resurs može biti knjiga, drugi servis kojeg se koristi, kolekcija određenih resursa, automobil, osoba, knjižnica itd. Stoga se resursi preslikavaju na skup entiteta, a ne entiteta koji se preslikavaju na resurse.

Identifikator resursa koristi se za jednoznačno označavanje resursa koji se koriste pri komunikaciji. Također predstavlja i referencu čiju semantiku i preslikavanje mora održavati nadležno tijelo koje omogućuje korištenje te reference. Također postoje statički identifikatori koji pokazuju na resurs čija se reprezentacija ne mijenja vremenom i dinamički identifikatori kojima se mogu promijeniti određeni parametri reprezentacije resursa.

2. Reprezentacija i metapodaci reprezentacije

REST obavlja radnje nad resursom kako bi mogao kreirati željeno stanje resursa, tj. kreirati njegovu reprezentaciju koja će se koristiti pri komunikaciji. Reprezentaciju predstavlja skup okteta koji je dobiven od nekog resursa te metapodataka reprezentacije koji opisuju kako su ti okteti formatirani da bi ih se moglo pravilo obraditi određenim programima. Metapodatke reprezentacije predstavljaju vrijednosti parova *ključ-vrijednost* koji u većini slučajeva odgovaraju HTTP metodama zaglavlja kojima se specificira datum kreiranja, vrijeme modifikacije, itd.

3. Kontrolni podaci

Kontrolni podaci predstavljaju one metode zaglavlja HTTP protokola u kojima se specificira rukovanje priručnom memorijom i one vrste reprezentacije nekog resursa koje klijent ili poslužitelj mogu prepoznati i obraditi. Također obuhvaća i metode kodiranja znakova i trenutni oblik

reprezentacije nekog resursa. Ukoliko jedan resurs ima više reprezentacija servis mora omogućiti proces pregovaranja oko posluživanja tražene reprezentacije klijentima.

**Elementi konektora**

Prema [11], definiraju se sljedeći tipovi konektora sadržanih u tablici 2.2. kojima se opisuju aktivnosti pristupa i prijenosa reprezentacije resursa.

Tablica 2.2. *REST elementi konektora*

| Element konektora | Primjeri |
|---|---|
| Klijent | Šalje zahtjeve poslužitelju i očekuje odgovore. |
| Poslužitelj | Odgovara na zahtjeve neovisno o tipu klijenta. |
| Priručna memorija | Locirana na sučelju klijenta, poslužitelja ili posrednicima. |
| Rješavač | Prevodi alfanumeričke identifikatore u numeričku mrežnu adresu. |
| Tunel | Prenosi komunikaciju preko vatrozida ili poveznika. |

Pomoću elemenata konektora omogućava se kreiranje apstraktnog sučelja za komunikaciju između komponenti. Postojanjem jedinstvenog sučelja preko kojega korisnici pristupaju servisu omogućava se promjena implementacije sučelja tj. pozadinskih radnji koje servis obavlja, a da se sučelje uopće ne promijeni. Također je moguće jednu veću informaciju prenijeti u više poziva pošto konektori pružaju tu mogućnost komponentama. Bitnu stavku REST arhitekture čini neovisnost zahtjeva (engl. *statelessness*) gdje svaki zahtjev sadržava dovoljnu količinu informacija da bi konektor mogao razumjeti i ispuniti traženi zahtjev. Trenutno primljeni zahtjev i njegova obrada ne ovise o prethodnoj povijesti zahtjeva od određene komponente. Time se postiže da:

- konektori ne moraju održavati praćenje aplikacijskog stanja,
- omogućena je paralelna obrada zahtjeva,
- omogućava posrednicima razumijevanje zahtjeva u odvojenosti od drugih zahtjeva te
- sve informacije koje bi se mogle ponovno iskoristiti u budućim odgovorima na iste zahtjeve trebaju biti naznačene u zahtjevima.

1. Klijent i poslužitelj

Klijent i poslužitelj konektori zaslužni su za komunikaciju u informacijskom sustavu. Klijent komponenta zadužena je za konstantno prozivanje poslužitelja kako bi dobila potraživane usluge ili podatke u obliku odgovora. Komponenta poslužitelja zadužena je konstantno na svojem sučelju očekivati zahtjeve te kako zahtjevi dolaze, tako ih obraditi, generirati odgovor i uputiti klijentu koji je inicirao zahtjev.

2. Priručna memorija

Konektor priručne memorije ima svrhu analizirati zahtjeve koje dolaze na sučelje poslužitelja ili klijenta u svrhu stavljanja odgovora na česte zahtjeve u priručnu memoriju. Time se pruža

otpornost poslužitelja na povećani promet pošto za svaki jednako formirani zahtjev ne mora generirati isti odgovor nego jednostavno već generirani odgovor za određeni zahtjev pročita iz priručne memorije te ga proslijedi kao novo generirani odgovor, a klijent ne mora upućivati novi zahtjev ako odgovor tog zahtjeva već ima u priručnoj memoriji.

3. Rješavač

Rješavač ima funkciju iz URI-ja prepoznati ime domene (autoriteta) te je uputiti DNS rješavaču koji će kao rezultat dati IP adresu tražene domene. Zatim program na tu adresu prosljeđuje ostatak putanje URI-ja te će se izvršiti formiranje zahtjeva.

4. Tunel

Tunel za funkciju ima prijenos komunikacije iz zatvorene mreže preko vatrozida ili poveznika do poslužitelja. Održava direktnu vezu između komponente klijenta i komponente poslužitelja sve dok se veza ne prekine.

**Komponente**

Prema [11], komponente unutar REST arhitekture podrazumijevaju one dijelove koji imaju ulogu kreiranja zahtjeva, prosljeđivanja zahtjeva i odgovora te točke koja prima zahtjeve i generira odgovore. Stoga komponente unutar REST arhitekture predstavljaju izvorni poslužitelj, posrednik, poveznik te korisnički agent.

1. Izvorni poslužitelj

Izvorni poslužitelj koristi konektor poslužitelja kako bi mogao nadzirati sve resurse kojim raspolaže te je ujedno i pravi izvor reprezentacija resursa koje pohranjuje. Ukoliko se zahtjevom mora mijenjati vrijednost resursa nužno je da bude krajnja točka koja će izvršiti navedenu izmjenu. Nije moguće uputiti konektor priručne memorije da promijeni vrijednost te da je proslijedi izvornom poslužitelju.

2. Poveznik

Poveznik je komponenta koja je nametnuta samom mrežom ili glavnim poslužiteljem kroz koju mora proći sav promet iz vanjskog svijeta do glavnog poslužitelja. Uobičajeno se koristi zbog zaštite glavnog poslužitelja, prevođenja podataka ili poboljšanja performansi.

3. Posrednik

Posrednik komponenta odabrana od korisničkog agenta koja može prosljeđivati zahtjeve korisničkog agenta i odgovore poslužitelja do određenog korisničkog agenta. Također može služiti

i kao točka pomoću koje se smanjuje opterećenje glavnog poslužitelja korištenjem priručne memorije iz koje će posluživati zahtjeve kao da dolaze od glavnog poslužitelja.

4. Korisnički agent

Korisnički agent koristi konektor klijenta za generiranje zahtjeva. Korisnički agent ujedno je i primatelj odgovora na zahtjev kojeg je inicirao. Najrašireniju skupinu korisničkih agenata predstavljaju razni web preglednici locirani na stolnim računalima ili mobilnim uređajima.

### 2.3.3. Stilovi

Prema [11], [13] postoji nekolicina mrežnih stilova arhitektura od kojih su najznačajniji stilovi protoka podataka, replikacijski stilovi, hijerarhijski stilovi, stilovi pokretnoga koda te stilovi ravnopravan-ravnopravnom. U nastavku će redom biti pojašnjeno nekoliko podstilova vezanih za određeni arhitekturalni stil. Glavni fokus stavljen je na hijerarhijske, replikacijske i stilove pokretnoga koda pošto REST predstavlja njihov hibrid.

**Stilovi protoka podataka (engl. *Data-flow Styles)***

Stil protoka podataka predstavlja arhitekturalni stil u kojem se nad podacima vrši uzastopna obrada i gdje su operacije neovisne jedna o drugoj [13]. Glavni elementi takvog stila jesu izvor, filter, cijevi i odredište.

Stil cijevi i filtera podrazumijeva postojanje međusobno neovisnih filtera i cijevi. U filterima se vrši transformacija podataka koje dobivaju sa svojih ulaza, a rezultat operacija prosljeđuju na svoje izlaze koji se zatim preko cijevi šalju do drugih filtera koji trebaju vršiti radnju nad podacima. Svaki filter inkrementalno čita podatke na svom ulazu te u isto vrijeme obrađuje podatke te rezultat operacije tada formira kao kontinuirani niz podataka što je dosta slično funkcioniranju privremenog spremnika (engl. *buffer*) ili reda kao strukture podataka koji koristi načelo FIFO (engl. *first-in, first-out*). Cijev i filter osiguravaju jednostavnost uporabe sustava, ponovno korištenje komponenti filtera uz uvjet da razumiju podatke koje čitaju s cijevi te mogu poboljšati odziv i performanse sustava. Nedostaci takvog sustava su mogući zastoji i smanjen odziv u slučaju primjene velikoga broja filtera koji trebaju vršiti nekakvu transformaciju nad podacima.

Stil jedinstvene cijevi i filtera, uz prethodno navedeno ponašanje stila cijevi i filtera, dodaje ograničenje nužnosti postojanja istog sučelja filtera u sustavu protoka podataka. Neovisnim razvojem različitih filtera koji se pridržavaju ograničenja istog sučelja moguće je kombiniranje filtera u svrhu stvaranja novih funkcionalnosti.

**Replikacijski stilovi (engl. *Replication Styles*)**

Stil repliciranog repozitorija omogućava pristup podacima i skalabilnost servisa tako što postoji više entiteta koji pružaju istu uslugu. Primjerice, velike organizacije poput Amazona imaju decentralizirane, tj. distribuirane računalne centre na kojima pohranjuju informacije o robama i korisnicima u svrhu pružanja otpornosti na kvarove ostalih računalnih centara i posluživanju većeg broja klijenata. U konačnici takav pristup omogućava i uravnoteženje opterećenja (engl. *load balancing*) cjelokupnog sustava te smanjenje vremena odziva na zahtjeve. Jedan primjer primjene stila repliciranog repozitorija očigledan je unutar Elasticsearch sustava koji ima mogućnost kreiranja kopija krhotina indeksa pomoću kojih se omogućava jednostavan oporavak od kvarova te distribuirano pretraživanje nad pohranjenim sadržajem.

Stil priručne memorije omogućava kreiranje privremene kopije nekog odgovora u svrhu posluživanja tog istog odgovora na iste buduće zahtjeve poboljšavajući pri tome vrijeme odziva servisa. Postoje dvije metode korištenja priručne memorije: lijena (pasivna) u kojoj se podaci stavljaju u priručnu memoriju pri prvoj potražnji za određenim podatkom te aktivna u kojoj se podaci stavljaju u priručnu memorije prije nego se zatraže zahtjevom. Navedeno se ostvaruje primjenom određenih algoritama koji iz normalnog režima korištenja servisa pokušavaju predvidjeti buduću potražnju za podacima te time smanjiti vrijeme odziva samog servisa, tj. poboljšati performanse.

**Hijerarhijski stilovi (engl. *Hierarchical Styles*)**

Stil klijent-poslužitelj najraširenija je vrsta arhitekture na globalnoj mreži razmjene informacija. Komponenta klijenta predstavlja računalo, web preglednik, mobilni uređaj ili servis koji zahtjeva informacije od drugog računala ili web servisa (komponenta poslužitelja) na standardiziran način. Zatim u ovisnosti o autentifikaciji i autorizaciji postavljenoj na strani poslužitelja, poslužitelj odlučuje hoće li ili neće proslijediti određene informacije kao odgovor na određeni zahtjev. Cilj postojanja dvije odvojene komponente omogućava njihov zaseban razvoj i skaliranje sve dok se sučelje preko kojega ostvaruju komunikaciju (API) ne promijeni.

Stil slojeviti sustav funkcionira tako da određeni sloj koristi usluge i sučelja nižeg sloja u svrhu pružanja svojih obradbenih mogućnosti sloju iznad gdje se zatim vrši složenija obrada i formatiranje tih podataka. Svaki sloj ograničen je na poznavanje samo onih slojeva koji se nalaze neposredno ispod i iznad njega. Takav pristup omogućava neovisan razvoj svakog sloja. Glavni nedostatak takvog stila povećanje je vremena odziva dok informacija prođe kroz svaki sloj te se dobije konačni rezultat svih operacija.

Slojeviti klijent-poslužitelj u stil klijent-poslužitelj dodaje posrednike (engl. *proxies*) i poveznike (engl. *gateways*). Posrednici služe da računala iz interne mreže šalju zahtjeve prema posredniku te zatim on vrši prosljeđivanje tih zahtjeva do odredišnih poslužitelja, a odredišni poslužitelj odgovor šalje na posrednika te posrednik vrši prosljeđivanje odgovora računalu koje je izdalo određeni zahtjev. Poveznik je računalo nametnuto od same mreže kroz koje mora prolaziti promet do interne mreže.

Stil udaljene sjednice sastoji se od činjenice da klijent započinje sjednicu koja se odražava na poslužitelju kojem pristupa. Klijent zatim može koristiti usluge raspoložive na odredišnom računalu kao da rukuje lokalnim računalom kojim pristupa usluzi. Takav stil pruža prednosti klijentu zato što ne mora imati veliku obradbenu moć pošto tu sposobnost ima udaljeno računalo kojem pristupa, a klijent dobiva rezultate zadanih operacija. Ostale prednosti takvog pristupa su lakši razvoj sučelja na strani poslužitelja te klijentske komponente. Nedostatak takvog pristupa jest otežana skalabilnost poslužitelja pošto se moraju spremati stanja sjednica korisnika koji koriste raspoložive usluge.

**Stilovi pokretnoga koda (engl. *Mobile Code Styles*)**

Obuhvaća stilove koji koriste mobilnost koda u svrhu mijenjanja lokacije izvršavanja obrade podataka ili mjesta dostave rezultata nekakve obrade nad podacima. S konceptom lokacije moguće je pri dizajniranju takvoga sustava razmotriti cijene koje su potrebne da se ostvari željena komunikacija te će tako komponente koje moraju koristiti određeni komunikacijski kanal imati višu cijenu korištenja od onih komponenti koje se nalaze na istoj lokaciji ili na nekoj manjoj udaljenosti pošto se korištenje komunikacijskih kanala plaća s vremenom njihova korištenja.

Stil kod na zahtjev obuhvaća takvu interakciju s poslužiteljem da poslužitelj klijentu pošalje resurse u obliku koji je neprepoznatljiv klijentu. Klijent zatim šalje novi zahtjev poslužitelju te potražuje kod koji treba pokrenuti kako bi mogao izvršiti obradu nad resursima dobivenih iz prethodnog zahtjeva. U nekim sustavima slanje resursa i koda potrebnog za interpretaciju obuhvaća se u jednom koraku. Sa stilom koda na zahtjev oslobađa se zauzeće poslužitelja s radnje obrade koda i resursa te se umjesto toga može posluživati veći broj klijenata. Takav pristup koristi se i danas pomoću JavaScript skriptnog jezika s kojim je korisnik u interakciji dok koristi posluženi sadržaj.

**Stilovi ravnopravan-ravnopravnom (engl. *Peer-to-Peer Styles*)**

Stil integracije temeljene na događajima klijentu omogućava prozivanje servisa bez potrebe poznavanja sučelja konektora. Takav pristup omogućava slabo vezivanje komponenti klijenta i

poslužitelja. Poslužitelj se u takvom stilu smatra skupom više komponenti. Komponente takvog sustava odašilju poruke koje su vezane za događaje te druge komponente mogu registrirati te događaje ako su zainteresirane za njihovu obradu. Obrada će se izvršiti sljedeći put kada sustav pozove registrirane komponente. Takav pristup omogućava proširivost pri dodavanju novih komponenti koje će slušati i registrirati događaje te nadogradnju komponenti pošto je svaka komponenta sustav za sebe, tj. nema strogog vezivanja komponenti.

C2 arhitekturalni stil zasnovan je na principima ponovnog korištenja i mogućnosti modificiranja komponenti zbog provođenja neovisnosti podloge na kojoj se razvijaju komponente. Neovisnost podloge omogućena je interakcijom stila temeljenog na događajima i slojevitog klijent-poslužitelj stila opisanih u ovome odjeljku. Razmjena zahtjeva i odgovora odvija se asinkronim porukama između slojeva kojima se razmjenjuju obavijesti stanja komponenti.

### 2.3.4. Ograničenja

Pošto REST predstavlja hibrid raznih stilova arhitekture iz kojih preuzima samo najbolje karakteristike kako bi pružno novi stil razvoja mrežnih arhitektura, njezin autor definirao je skup ograničenja koja se primjenjuju na elemente arhitekture. Prema [11], [12] ograničenja koja se nameću na tako formiranu arhitekturu su: klijent-poslužitelj, neovisnost zahtjeva, priručna memorija, jedinstveno sučelje, slojeviti sustav te kod na zahtjev.

**Ograničenje klijent-poslužitelj**

U arhitekturalni stil uvodi princip odvajanja odgovornosti (engl. *separation of concerns*). Omogućava komponenti klijenta da neovisno o poslužitelju razvija prikaz korisničkog sučelja i implementaciju načina komunikacije s poslužiteljem što omogućava prenosivost sučelja između više platformi. Poslužitelj se ne mora fokusirati na vezivanje podataka koje pohranjuje i korisničkog sučelja koje bi kreirao. Točnije, poslužiteljev cilj jest pohrana podatka i pružanje usluga klijentima koji ih zahtijevaju povećavajući time skalabilnost poslužitelja.

**Ograničenje neovisnosti zahtjeva**

Ovim ograničenjem u arhitekturu se unosi pravilo da ne smije postojati sjednica u komunikaciji između klijenta i poslužitelja. Ako bi stanje sjednica bilo prisutno na poslužitelju, zahtjevi za obradbenom moći poslužitelja bili bi veći, a broj klijenata koje bi mogli poslužiti bio bi manji nego kada ne bi bilo održavanja stanja sjednice. Također bi bilo teže izvršiti distribuiranost sustava te uravnoteženje opterećenja zbog prijenosa stanja sjednica između poslužitelja. Pomoću neovisnosti zahtjeva stavlja se naglasak na postojanje svih potrebnih informacija unutar zahtjeva kako bi poslužitelj mogao ispravno obraditi zahtjev te ako se želi, klijentska strana može uvesti praćenje

sjednice za svrhu navigacije po prethodno primljenim stanjima. Prednosti takvog sustava su poboljšanja u vidljivosti, pouzdanosti i proširivosti sustava. Vidljivosti zato što poslužitelj ne mora tražiti kontekst u kojem se pojavio zahtjev u ovisnosti o prethodnim zahtjevima, proširivosti zbog toga što ne mora spremati stanje sjednice za svakoga korisnika, a pouzdanosti zbog jednostavnijeg oporavka od kvarova. Nedostatak takvog pristupa je prosječno veća količina mrežnog prometa potrebna da se ostvari jedan ciklus zahtjev-odgovor.

**Ograničenje priručne memorije**

Ograničenje priručne memorije u mrežni sustav uvodi sposobnost spremanja odgovora na zahtjeve. Odluka o implicitnoj ili eksplicitnoj uporabi ili zabrani uporabe priručne memorije stoji na poslužitelju prilikom kreiranja odgovora na zahtjev. Ako poslužitelj odluči da će neki odgovor moći biti spremljen u priručnu memoriju tada će taj odgovor moći biti korišten kao odgovor na identičan zahtjev. Odgovori mogu biti pohranjeni u poslužitelju, posrednicima te klijentu. Ako su odgovori pohranjeni u memoriji klijenta, moguće je u potpunosti eliminirati promet prema poslužitelju ako klijent napravi novi identičan zahtjev kojeg je prvo uputio poslužitelju. U priručnu memoriju nije uputno spremati odgovore čiji se sadržaj često mijenja s vremenom tj. priručna memorija korisnija je za statične sadržaje. Priručnom memorijom može se poboljšati odziv na zahtjev i proširivost pošto se odgovori mogu pohranjivati na posrednicima u slučaju većeg zauzeća glavnog poslužitelja.

**Ograničenje jedinstvenog sučelja (engl. *Uniform Interface*)**

Jedinstveno sučelje predstavlja sloj apstrakcije iza kojega se definira funkcionalnost servisa. Takav pristup omogućava neovisan razvoj implementacije sučelja od apstrakcijskog sloja preko kojega se vrši komunikacija s klijentom tj. primanje zahtjeva za obradu. Nedostatak takvog pristupa je prisiljenost na uporabu standardiziranih metoda i tipova podataka ponuđenih HTTP protokolom zbog toga što se ne može izvršiti stopostotna prilagodba za svaku primjenu. Prednost jedinstvenog sučelja prepoznata je u jednostavnijoj arhitekturi cjelokupnog sustava, no kako bi se ostvarila prava jedinstvenost sučelja, potrebno je na sučelje primijeniti četiri ograničenja:

- Identifikacija resursa: Identifikacija resursa unutar REST-a treba se odvijati korištenjem URI standarda.
- Manipulacija resursa pomoću reprezentacija: Klijent pomoću reprezentacije i metapodataka koje dobije kao odgovor na zahtjev ima mogućnost dodatnih upita na isti resurs u svrhu izmjene ili brisanja tog resursa. Ta funkcionalnost trebala bi biti zabranjena „vanjskom svijetu“ ili strogo kontrolirana na samo uski skup resursa nad kojima klijent može imati ovlasti modificiranja ili brisanja.

- Samoopisne poruke: Svaki zahtjev mora imati sposobnost da u sebi ukomponira sve potrebne elemente kako bi poslužitelj znao razumjeti stavke zahtjeva te formatirati pravilan odgovor na takav zahtjev. Samoopisnost se postiže definiranjem vrste zahtjeva HTTP metodom i tipom sadržaja odgovora kojeg klijent prihvaća definiranjem MIME (engl. *Multipurpose Internet Mail Extension*) tipa. MIME definicija sadržaja stavlja se pod *Content-Type* metodu zaglavlja unutar HTTP poruke, a sastoji se od tipa i podtipa sadržaja odvojenih kosom crtom. Prema [35] najznačajniji tipovi su *text, image, application,* no postoje i ostali tipovi. Najznačajnije podtipove predstavljaju *json, xml* i *html*, a potpuna lista tipova i podtipova raspoloživa je na [36].
- Hipermedija kao pokretač aplikacijskog stanja (engl. *Hypermedia as the Engine of Application State*; HATEOS): U svakom odgovoru poslužitelja na zahtjev trebali bi se dati primjeri pristupanja drugim resursima koje sučelje podržava. Točnije, korisniku se trebaju prikazivati primjeri navigacije po implementiranom sučelju poslužitelja bez njegova prethodna znanja o toj navigaciji.

**Ograničenje slojevitog sustava (engl. *Layered System*)**

Omogućava komponentama sustava da se razvijaju neovisno o drugim komponentama uz uvjet da zadržavaju ista sučelja preko kojih komuniciraju. Takav pristup omogućava komponentama da vide samo do sljedeće komponente što smanjuje složenost sustava. Nedostatak takvog pristupa je povećanje vremena odziva da zahtjev prođe kroz sve posrednike i poveznike te da dođe to glavnog poslužitelja, no taj se nedostatak može djelomično ublažiti korištenjem ograničenja priručne memorije.

**Ograničenje koda na zahtjev (engl. *Code-on-Demand*)**

Predstavlja jedino opcionalno ograničenje unutar REST arhitekture kojim se omogućava poslužitelju da uz reprezentaciju pošalje i kod kojeg klijent može koristiti za proširivanje svojih funkcionalnosti. Korištenjem tog pristupa smanjuje se opterećenje poslužitelja pošto ne mora izvršiti kod čiji bi rezultat zatim poslao klijentu. Ograničenje je opcionalno zato što se preuzimanje raznih skripti može zabraniti korištenjem vatrozida i ostalih sigurnosnih sustava dok neki sustav koji ne zabranjuje preuzimanje skripti, može implementirati sustav koji podržava ograničenje koda na zahtjev.

## 2.4. Prijenosni formati

Pri komunikaciji heterogenih sustava nužno je postaviti određene standarde kako bi se mogla nesmetano odvijati komunikacija između servisa međusobno ili između klijenta i servisa. U protivnom nastaju „neredi“ gdje bi svaka jača tvrtka donosila svoje *kvazi* standarde te bi klijenti koji bi surađivali s određenom tvrtkom morali razvijati posebnu programsku podršku kako bi mogli obavljati aktivnosti svoga poslovanja. U tu svrhu nadležna tijela Interneta poput IETF-a i W3C-a imaju funkciju donošenja standarda i specifikacija kojih bi se bilo dobro pridržavati kako bi se omogućila kompatibilnost između heterogenih sustava. U svrhu prijenosa podataka između servisa ili klijenta i servisa koriste se različiti prijenosni formati kao što su XML, JSON, XDR, NDJSON, BSON, SMILE, CBOR, YAML, CSV i drugi. Od navedenih, najširu uporabu imaju XML format te JSON format čiji će koncepti biti pojašnjeni u narednim odjeljcima.

### 2.4.1. XML

Skraćenica XML dolazi iz engleskog govornog područja od riječi *Extensible Markup Language* što u prijevodu znači proširivi označni jezik. Proširivo svojstvo XML jezika dolazi od činjenice da jednom implementirana funkcionalnost korištenja XML odgovora na zahtjev ostaje stalna sve dok se ne promijeni format odgovora nad kojim je implementirana funkcionalnost. To znači da servis kontinuirano može dodavati nove elemente u odgovore s očuvanjem prethodno implementiranih, a da korisnici odgovora ne moraju dorađivati svoj program, servis ili web stranicu ako su zadovoljni trenutnom funkcionalnošću svojih programa, servisa ili web stranica. XML jezik ne vrši nikakvu funkciju ili manipulaciju nad podacima, tj. služi samo za transport samoopisivih informacija. Također je neovisan o programskom ili sklopovskom okruženju zbog toga što sve podatke sadržava u tekstualnom formatu što značajno pojednostavljuje dijeljenje sadržaja i otpornost na promjene sustava koji međusobno komuniciraju. Glavnu razliku između XML i HTML označnih jezika čini XML-ova fokusiranost na opisivanje podataka i njihov prijenos dok se HTML bazira na prikazivanju tih podataka. Za svrhu povezivanja dva jezika može se koristiti skriptni jezik JavaScript koji može vršiti pretvaranje reprezentacije u HTML dokument ili može enkapsulirati podatke u XML dobivene iz HTML dokumenta. Svojstvo samoopisnosti XML jezika dolazi od činjenice da sadržava gotovo beskonačno mnogo mogućih oznaka koje se koriste za označavanje podataka. Te oznake nisu propisane standardom kao što je riječ u HTML-u nego su proizvoljne, tj. autor servisa može dodijeliti koje god ime želi ali se pri tome mora pridržavati određenih pravila pri imenovanju elemenata. Ograničenja koja se stavljaju na imenovanje elemenata dosta su slična pravilima ostalih programskih jezika od kojih su: ime elementa ne smije

započinjati brojevima, smiju započinjati slovima, u svome imenu ne smiju sadržavati prazna mjesta, smiju sadržavati '_' , brojeve te ne smiju sadržavati riječ xml. Za svaki XML kompatibilan dokument poželjno je da u prvoj liniji dokumenta ima naznačenu inačicu XML jezika koja je korištena u dokumentu te koje je kodiranje znakova korišteno u dokumentu. Svaki XML dokument mora sadržavati jedan korijenski čvor (engl. *root node*). Unutar korijenskog čvora može postojati više istoimenih elemenata djece koji također mogu imati više svojih elemenata djece, tj. stvara se hijerarhija unutar dokumenta. Ta hijerarhija XML dokumenta postaje vidljiva DOM prevoditelju (engl. *Data Object Model parser*) prilikom pozivanja navedenog prevoditelja. Zatim se jednostavnim metodama vrši navigacija po kreiranom dokumentu. Elementi unutar XML dokumenta moraju biti pravilno ugniježđeni te pravilno zatvoreni kako je prikazano u nastavku. Pravilo zatvaranja elemenata, prema [14] nalaže da moraju koristiti isto ime koje je korišteno za otvaranje elementa, a također postoje i takozvani samozatvarajući elementi koji smiju sadržavati samo atribute kako je pokazano u nastavku u primjeru 2.1.:

Primjer 2.1. *Elementi XML-a*

| | |
|---|---|
| <fakultet> FERIT </fakultet> | Ispravno |
| <fakultet> FERIT </Fakultet> | Neispravno |
| <fakultet naziv="FERIT" /> | Samozatvarajući element |

Atributi unutar XML dokumenta moraju biti pisani malim slovima i moraju imati pridruženu vrijednost uokvirenu unutar para jednostrukih ili dvostrukih navodnika. Atributi bi se trebali izbjegavati pri definiranju sadržaja dokumenta zbog nemogućnosti ugnježđivanja dodatnih elemenata i nefleksibilnosti jednostavnog proširenja te bi se trebali koristiti samo za pohranu metapodataka. U XML dokument moguće je uvesti područje imenika korištenjem riječi xmlns te ':' iza kojeg se definira prefiks koji će se koristiti prije imena elemenata. Područje imenika unosi se u dokument kako bi se znala razlika između dokumenata s različitom svrhom, a istim nazivima elemenata. Područje imenika može se definirati u korijenskom elementu ili u elementu u kojem je potrebno uvesti područje imenika. U slučaju da je u sadržaj elemenata XML dokumenta potrebno dodati tekst koji je formatiran HTML oznakama ili koristiti zabranjene znakove, potrebno je takav tekst označiti u bloku „<![CDATA[proizvoljni_tekst]]>". Ograničenje koje se stavlja na taj blok, tj. na tekst koji sadržava, jest nemogućnost sadržavanja znakova „]]>" koji terminiraju navedenu oznaku. Oznaka CDATA služi za održavanje ispravnosti XML dokumenta jer bi u protivnom prevoditelj obavijestio o pogrešno formatiranom dokumentu. U slučaju kada se postigne željena forma XML dokumenta te njena funkcionalnost, programerima je omogućeno kreiranje definicije dokumenta koji bi služio za razmjenu informacija. Takav dokument naziva se XML definicija tipa

dokumenta (engl. *XML Document Type Definition;* XML DTD). U tom dokumentu definiraju se ograničenja na dozvoljene elemente i atribute koje svaki XML dokument mora poštovati ako unese XML DTD dokument u svoju definiciju.

U kratkom primjeru 2.2. u nastavku bit će izložen jednostavan XML dokument koji prati načela ovoga odjeljka, ali ne podrazumijeva ispravnost navedenih informacija u sadržaju elemenata:

Primjer 2.2. *Primjer XML dokumenta*

```
<?xml version="1.0" encoding="utf-8"?>
<korijen
        xmlns:f="http://fakultet"
        xmlns:k="http://knjiznica">
        <f:fakultet id="1">
                <f:naziv>FERIT</f:naziv>
                <f:broj_predavaona>45</f:broj_predavaona>
                <f:povrsina>1000</f:povrsina>
                <f:broj_prozora>60</f:broj_prozora>
                <f:ulica> Ulica Kneza Trpimira</f:ulica>
        </f:fakultet>
        <k:knjiznica id="2">
                <k:naziv>GISKO</k:naziv>
                <k:predavaona>5</k:predavaona>
                <k:povrsina>900</k:povrsina>
                <k:broj_prozora>30</k:broj_prozora>
                <k:ulica>Europska avenija</k:ulica>
        </k:knjiznica>
</korijen>
```

### 2.4.2. JSON

Skraćenica JSON dolazi iz engleskog govornog područja od riječi *JavaScript Object Notation* što u prijevodu znači označavanje objekata po uzoru na JavaScript. JSON kao i XML služi za pohranu i razmjenu informacija između web servisa u tekstualnom obliku te je također neovisan o programskom jeziku i računalnoj platformi. Zbog jednostavnosti implementiranja, malog zauzeća računalnih resursa te pretvaranja u i iz JavaScript objekta preferiran je format za kreiranje web servisa. Za razumijevanje JSON formata nužno je poznavanje sintakse objekta i polja. Prema [15], svaki objekt otvoren je lijevom vitičastom zagradom '{' zatvoren desnom vitičastom zagradom '}', a polje je otvoreno lijevom uglatom zagradom '[' i zatvoreno desnom uglatom zagradom ']'. Objekti sadrže elemente u obliku parova *ključ-vrijednost* kojih može biti 0 ili više. Svi ključevi unutar JSON dokumenta moraju biti tekstualnog oblika unutar para dvostrukih navodnika. U cijelom dokumentu moguće je imati više ključeva s istim identifikatorom, ali moraju biti unutar drugog

objekta ili polja kako prevoditelj ne bi prepoznao samo zadnje ponovljeni identifikator (neki prevoditelji će pojaviti grešku ako se više identifikatora nalazi na istoj razini hijerarhije). Vrijednost koju ključ može poprimiti prema [15] može biti drugi objekt, polje, tekstualni niz (engl. *string*), broj (cijeli broj ili broj s pomičnom točkom), *true*, *false* i *null*. Polje unutar JSON objekta obilježava par uglatih zagrada te mogućnost pohrane drugih objekata ili vrijednosti kojih može biti 0 ili više koji se asociraju na ključ polja. Parove *ključ-vrijednost* nužno je odvajati jednim dvotočjem ':'. Za dodavanje novih *ključ-vrijednost* parova u postojeći JSON objekt, dodavanje novih objekata u JSON objekt ili polje te za dodavanje niza parova *ključ-vrijednost* unutar polja, nužno je koristiti oznaku zareza ',' kako bi se elementi odvojili. Kao i XML, JSON omogućava ugnježđivanje elemenata no implementacija prevoditelja može ograničiti najveću dopuštenu razinu ugnježđivanja. Također, prema specifikaciji u [15] implementacija prevoditelja može ograničiti duljinu teksta koju će prevoditelj prihvatiti (cjelokupnog JSON dokumenta kao broj znakova potrebnih da se dokument definira), može ograničiti raspon i preciznost zapisa brojeva te može ograničiti duljinu i znakove koji se smiju koristiti pri definiranju tekstualnih nizova. U primjeru 2.3. bit će prikazan primjer 2.2. ali u JSON notaciji, a u primjeru 2.4. bit će prikazano ugnježđivanje objekata.

Primjer 2.3. *Primjer JSON dokumenta kao rezultat transformacije primjera 2.2.*

```
{
  "fakultet":{
    "id":1,
    "naziv":"FERIT",
    "broj_predavaona":45,
    "povrsina":1000,
    "broj_prozora":60,
    "ulica":"Ulica Kneza Trpimira"
  },
  "knjiznica":{
    "id":2,
    "naziv":"GISKO",
    "broj_predavaona":5,
    "povrsina":900,
    "broj_prozora":30,
    "ulica":"Europska Avenija"
  }
}
```

Primjer 2.4. *Primjer korištenja polja i razine ugnježđivanja u JSON dokumentu*

```
{
  "automobilska_kuca":"Ferrari",
  "korisnici":[
    {
      "ime":"Ivo",
      "prezime":"Ivic",
      "automobili":[
        {
          "naziv":"Ferrari",
          "model":"LaFerrari"
        },
        {
          "naziv":"Ferrari",
          "model":"F12"
        }
      ]
    }
  ]
}
```

Kao što je vidljivo, za definiranje primjera 2.3. potrebno je značajno manje znakova nego u ovisnosti o primjeru 2.2. U primjeru 2.4. prikazana je jednostavna hijerarhija objekata i polja u JSON dokumentu počevši od polja korisnika koje zatim sadržava objekt koji sadrži informacije o kupcu i automobilima koje posjeduje u obliku polja koje sadrži informacije o dva tipa automobila u vlasništvu korisnika. JSON format također je korišten i za odgovore servisa korištenog u praktičnom dijelu ovoga rada, a nalazi se u tablicama 3.1. – 3.6.

## 2.5. RESTful servisi

Da bi neki web servis postao RESTful servis mora poštovati ograničenja definirana u odjeljku 2.3.4. RESTful servisi koriste se na globalnoj mreži za razmjenu informacija s ciljem dohvaćanja informacija ili obavljanja poslovne funkcionalnosti. Web servise mogu prozivati različiti tipovi uređaja ili ostali web servisi kako bi koristili već gotov rad za dio problema kojeg rješavaju.

### 2.5.1. Najbolje prakse

Kako bi se olakšao inicijalni razvoj RESTful servisa mnogi programeri su nakon godina iskustva pri razvoju RESTful web servisa odlučili uokviriti stečeno znanje u svrhu pomaganja novim programerima. U narednom tekstu prema [16], [17], [18], [19] pojasnit će se principi kojih bi se trebalo pridržavati pri razvoju RESTful web servisa.

- Poželjno je koristiti HTTPS inačicu HTTP protokola za komercijalne svrhe zbog pružanja enkripcije SSL ili TLS protokolom koji koriste kriptografiju javnim ključem [34]. HTTPS omogućava enkripciju putanje URI-ja, HTTP zaglavlja te tijela HTTP poruke čime se osigurava povjerljivost poruka koje se šalju između klijenta i poslužitelja.
- Imenovanje URI-ja trebalo bi se izvršavati malim slovima. Moguće je koristiti povlake ili donje crte, no uputnije je koristiti povlake. Poželjno je pridržavati se odabranog načina imenovanja kroz cjelokupni razvoj servisa.
- Imena resursa koji označavaju kolekciju resursa trebala bi biti pisana u množini, a pojedinačnih dokumenata u jednini što uvodi razinu intuitivnosti. Da bi se uveo hijerarhijski odnos u URI, potrebno je koristiti kosu crtu '/'. Za upravljače uputno je koristiti glagol ili frazu koja opisuje što upravljač radi.
- Svaki resurs kojem bi se pristupalo pomoću servisa trebao bi biti u URI-ju zapisan kao imenica, a HTTP metoda kojom bi se pristupalo resursu trebala bi biti izražena kao glagol
- Programer se mora ograničiti na one metode koje su raspoložive korištenjem HTTP protokola, tj. ne može kreirati vlastite metode za manipulaciju resursa.

- Da bi web servis bio RESTful web servis, na HTTP protokolu mora koristiti ugrađene HTTP metode – GET, PUT, POST, DELETE.
- Poželjno je da servis kao odgovor podržava vraćanje više tipova reprezentacija nekog resursa. Uobičajeno je reprezentacija servisa u JSON ili XML formatu. Reprezentaciju koju klijent zahtjeva moguće je specificirati u zaglavlju HTTP zahtjeva, tj. unutar *Accept* polja. Također je moguće omogućiti klijentu da zahtijevanu reprezentaciju naznači na kraju URI-ja. Također se za upućivanje klijenta ili poslužitelja u tip podatka koji se nalazi unutar tijela HTTP poruke koristi element zaglavlja *Content-Type* unutar kojeg se specificira MIME tip sadržaja.
- Korisniku se treba omogućiti navigacija kroz servis koristeći hipermediju, a korisnik dolazi u dodir sa sjedištem servisa pronađenog pomoću Internet tražilice
- U ovisnosti o tome kakva vrsta RESTful servisa se razvija, potrebno je omogućiti autentifikaciju i autorizaciju korisnika.
- Potrebno je što je više moguće koristiti priručnu memoriju za resurse koji se često ne mijenjaju jer rezultira poboljšanim vremenom odziva te mogućnosti posluživanja više upita. Može postojati unutar poslužitelja, klijenta ili na računalima između njih.
- Poželjno je omogućiti *straničenje* (engl. *pagination*) tj. omogućiti klijentu da izvrši restrikciju broja rezultata koje može dobiti kao odgovor servisa. U konačnici rezultira bržim odzivom zato što je potrebno prenijeti manju količinu podataka.
- Nalaže se potreba za korištenjem HTTP statusnih poruka kako bi se klijenta uputilo u stanje njegovog zahtjeva upućenog na web servis. Uz općenite statusne poruke poželjno je u tijelo takvog odgovora dodati kratko pojašnjenje poduzete radnje servisa. U tablici 2.3. prikazani su standardni statusni kodovi koji se koriste za određivanje stanja web servisa.

Tablica 2.3. *Primjer HTTP statusnih kodova*

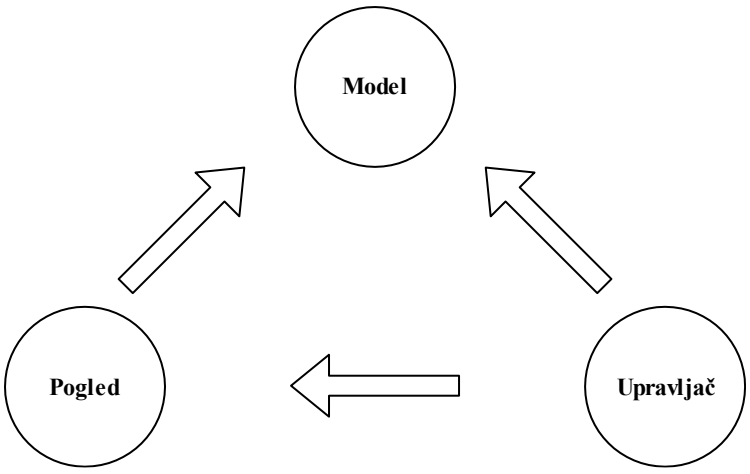
| Statusni kod | Značenje |
|---|---|
| 200 | Zahtjev je primljen i obrađen te se u tijelu HTTP metode nalaze tražene informacije. |
| 201 | Kreiran je specificirani resurs te se lokacija tog resursa nalazi u zaglavlju. |
| 202 | Ima isto značenje kao i statusni kod 200 ali se koristi prilikom asinkronog poziva servisa. |
| 303 | Traženi resurs je promijenio lokaciju te bi nova lokacija trebala biti uključena u odgovor. |
| 400 | Loše formiran zahtjev. Klijent mora ponovno formirati zahtjev. |
| 401 | Korisnik nije autoriziran tj. nema ovlasti pristupa traženom resursu. |
| 403 | Korisnik je uspješno autentificiran, no nema pristup traženom resursu. |
| 404 | Resurs kojeg klijent zahtjeva ne postoji ili klijentu nije dozvoljen pristup resursu. |
| 500 | Pogreška rada poslužitelja. Poslužitelj će možda izložiti informaciju o pogrešci. |

## 2.6. Oblikovni obrazac MVC

MVC oblikovni obrazac predstavlja razvojni stil u kojem se prilikom rješavanja jednog problema taj problem razdvaja na 3 manja problema. Prema [20], svrha MVC oblikovnog obrasca je izvršiti tzv. razdvajanje odgovornosti s ciljem smanjenja složenosti razvoja nekog programa ili web servisa. Ukoliko bi se razvijao sustav u kojemu se istovremeno razvija i poslovna logika i prikaz korisničkog sučelja, drastično se povećava složenost rješavanja problema ako bi se trebala mijenjati ili poslovna logika ili prikaz korisničkog sučelja pošto bi te dvije komponente bile strogo integrirane. Olakšava pisanje koda, uklanjanje grešaka u kodu, unošenje automatiziranih testova te održavanje samog programa što u konačnici ostvaruje odgovornost na dio problema i omogućava realizaciju čak i naprednijih funkcionalnosti za neki program ili servis. Olakšava istovremeni razvoj programa tako da svaki tim razvija jedan njegov dio te omogućava ponovno korištenje koda pri realizaciji drugačijih programa koji trebaju sličnu funkcionalnost. Stoga se model u MVC oblikovnom obrascu oblikuje kao samostalna jedinica koju pozivaju pogled i upravljač. Glavne 3 komponente MVC oblikovnog obrasca pri razdvajanju složenog problema su:

- model podataka i poslovne logike nad tim podacima (engl. *model*),
- upravljač (engl. *controller*), te
- pogled (engl. *view*).

Na slici 2.4., prema [20], nalazi se prikaz ovisnosti triju navedenih komponenti.




Slika 2.4. *Prikaz komponenti MVC modela*

Dužnost modela jest implementirati svu poslovnu logiku i svaku manipulaciju koja bi se mogla dogoditi nad podacima tijekom rada programa. Model treba biti takav da postoji kao jedinka koja ne zna za postojanje ostale dvije komponente – pogleda i upravljača koji prozivaju model s ciljem

izvršavanja radnji nad podacima te dobivanja rezultata tih radnji, tj. treba omogućiti svojevrsni API preko kojega se pristupa podacima. Model ne vrši direktnu komunikaciju s pogledom ako se dogodi promjena nad podacima, no može implementirati sustav odašiljanja poruka u ovisnosti u kojem je stanju te zatim pogled ako je programiran da prepozna te poruke, može od upravljača zahtijevati osvježene informacije.

Pogled je zadužen za posluživanje HTML stranica koje mogu sadržavati *Razor* označni jezik specifičan za .NET okruženje jer omogućava implementiranje C# koda vezanog za prezentaciju podataka korisniku. Implementacija omogućava kombiniranje koda na poslužitelju i koda u pogledu za prezentiranje sadržaja što je dosta slično JavaScript skriptnom jeziku koji se isto može ugrađivati u HTML stranice kako bi se na klijentskoj strani omogućila dinamičnost i interaktivnost prilikom korištenja stranice.

Upravljač predstavlja nekakvu vrstu sučelja koje se nalazi na granici domene vanjskog svijeta preko kojega klijenti žele pristupati programu ili servisu. Omogućava prihvat korisničkih zahtjeva, odabir odgovarajućeg modela podataka, posluživanje potrebnih podataka pogledu te posluživanje pogleda.

Osnovni princip razdvajanja odgovornosti može se primjenjivati i na razvoj web api servisa koji nije strogo vezan za poglede poput MVC-a nego bi umjesto pogleda, kao rezultate zahtjeva, slao podatke strukturirane u odgovoru u XML ili JSON formatu.

Prema [21], prednosti korištenja MVC razvojnog obrasca su:

- istovremeni rad programera na modelima, upravljačima i pogledima potrebnim za rad programa,
- MVC oblikovni obrazac omogućava logičku grupaciju radnji u upravljače te grupaciju pogleda za određeni model,
- modeli mogu imati više pogleda,

a nedostaci korištenja MVC razvojnog obrasca su:

- navigacija kroz okruženje može postati složena zbog novih razina apstrakcije prilikom razvoja te
- potrebno konstantno usavršavanje programera.

## 2.7. Razvoj RESTful servisa u .NET okviru

U današnje vrijeme postoji izrazita potreba za servisima dostupnih širokoj javnosti. Za razvoj RESTful servisa moguće je koristiti programske jezike poput Jave, C#, Python itd. U nastavku će se kroz korake navesti kako izgleda razvoj jednostavnog RESTful servisa unutar .NET okruženja pošto je u tom okruženju kreirano rješenje za pretragu vijesti koje je opisano 3. poglavljem.

1. Određivanje zadatka kojeg se treba riješiti.
2. Skicirati ili modelirati traženo ponašanje zadanog problema.
3. Prikupiti potrebna znanja i vještine koje će biti potrebne za realizaciju rješenja problema.
4. Instalirati razvojni alat Visual Studio.
5. Pokrenuti alat Visual Studio.
6. Odabrati *File->New->Project*. U novootvorenom prozoru potrebno je otići na kartice redom: *Templates -> Visual* C# *-> Web* te u sredini prozora odabrati *ASP.NET Web Application (.NET Framework)*. Zatim je potrebno definirati ime projekta te pritisnuti OK. U novootvorenom prozoru potrebno je odabrati iz prozora predložaka *Web API* te staviti kvačicu na *Add Unit Tests*.
7. Zatim će ugrađeni sustav generirati projekt i potrebne osnovne datoteke, upravljače, konfiguracije, klase.
8. U generirani projekt potrebno je definirati vlastite klase, sučelja, metode i upravljače. Ti elementi trebaju biti oblikovani tako da pruže potporu cjelokupnoj poslovnoj logici projekta. Poslovnu logiku potrebno je u potpunosti odvojiti od upravljača (ne i pogleda zato što je kreiran projekt tipa Web API koji nije strogo vezan za poglede, ali može ih imati). U klasama se definiraju podaci koji su od važnosti za cijeli servis koji se kreira. Metode trebaju manipulirati nad definiranim podacima bilo to u komunikaciji s drugim web servisom, bazom podataka ili nešto slično. Također, potrebno je vratiti rezultate obrade, pružiti stabilnu podlogu za rukovanje iznimkama te omogućiti pozivanje tih metoda kako bi se mogle koristiti iz drugih klasa, upravljača, metoda, tj. za razvoj poslovne logike. U upravljačima je potrebno definirati kojom HTTP metodom rukuju te koju će definiranu metodu za obradu podataka pozvati da dobiju rezultat određene operacije. Nakon dobivanja rezultata obrade, potrebno je u upravljaču formatirati odgovor kojeg će dobiti klijent na poslani zahtjev.
9. U slučaju da se iz web servisa želi vraćati samo JSON format odgovora, potrebno je unutar mape *App_Start* odabrati klasu *WebApiConfig.cs* te u njoj definirati da će se u globalnoj konfiguraciji koristiti samo JSON format. U toj klasi također je moguće definirati i vlastite rute upravljača koje će biti preslikane iz URL-a prilikom rada web servisa.

10. U klasi *Tests* moguće je definirati metode kojima bi se ispitivala poslovna logika bez potrebe za pokretanjem cjelokupnog servisa. Služi za testiranje i otklanjanje mogućih problema koji mogu nastati pri razvoju.
11. Testiranje cjelokupnog servisa koristeći sve implementirane funkcionalnosti.
12. Izrada detaljne dokumentacije servisa.
13. Objava servisa u poslovne svrhe.
14. Održavanje te razvoj mogućih nadogradnji na postojeći servis i dokumentiranje novih sposobnosti servisa.

# 3. PROGRAMSKO RJEŠENJE ZA PRETRAGU VIJESTI - *QuickSearch*

Za realizaciju praktičnog dijela završnog rada i razumijevanje rješenja *QuickSearcha*, potrebno je pružiti pogled u korištene platforme i alate te izložiti zahtjeve za sustav kako bi cjelokupni projekt mogao funkcionirati, a bit će opisani potpoglavljem 3.1. Potpoglavljem 3.2. pojasnit će komponente kreiranog web servisa, a u potpoglavlju 3.3. prikazat će se i komentirati zahtjevi i odgovori web servisa.

## 3.1. Zahtjevi na sustav, opis platformi, alata i tehnologija

Za realizaciju *QuickSearcha* korišteno je razvojno okruženje Visual Studio, C# programski jezik, .NET okvir, paket NEST za komunikaciju sa servisom Elasticsearcha iz razvojnog okuženja Visual Studio, Elasticsearcha koji se pokreće kao servis na računalu te alat Postman za jednostavno upućivanje upita prema Elasticserach servisu ili razvijenom servisu unutar Visual Studija. Također je bitno i razumijevanje RSS standarda za grupaciju vijesti. U nastavku će biti kratko opisani spomenuti alati i tehnologije te će se naglasak staviti na Elasticsearch te njegove osnovne metode pretrage koje su korištene u projektu.

### 3.1.1. RSS

RSS je skraćenica koja dolazi iz engleskog govornog područja od riječi *Really Simple Syndication*, a što u kontekstualnom značenju prenosi misao „stvarno jednostavno sjedinjenje“. RSS služi za sjedinjenje vijesti koje objavljuje neko web sjedište u svrhu povećanja preglednosti objavljenih vijesti i povećanja prometa prema sjedištu. Postoje različiti programi koji svojim korisnicima omogućavaju dodavanje izvora vijesti s ciljem da korisnik ne mora pretraživati web sjedišta da pronađe vijesti koje ga zanimaju, nego se na centraliziranom mjestu omogućava pregled naslova i odabir poveznice na željenu vijesti. Također se može preuzeti javno dostupan RSS dokument s ciljem razvoja vlastitog programa za pretragu i pohranu vijesti. Prema [22], inačicu RSS 0.9 razvila je tvrtka UserLand Software 1997. godine. RSS dokument pisan je XML-om te korijenski element tog dokumenta mora sadržavati atribut u kojem je zabilježena inačica RSS-a koja se koristi u dokumentu. Pod elementom „kanal“ (engl. *channel*), prema [23], specifikacijom propisane oznake koje taj element mora sadržavati su: naslov sjedišta koje objavljuje vijest, poveznice na sjedište te kratkog opisa sjedišta. Proizvoljne oznake elementa kanal predstavljaju datum objave dokumenta, zadnje vrijeme izmjene sadržaja, kategoriju itd. RSS dokument uobičajeno se sastoji od 15 elemenata, tj. element RSS sastoji se od 15 elemenata djece od kojih svaki predstavlja zasebno objavljenu vijest (engl. *item*). Da bi se zadovoljila forma RSS specifikacije svaki element vijesti

mora imati ili naslov ili opis koji može sadržavati tekst formatiran pomoću HTML-a kako bi služio za prikaz na nekom sjedištu ili posebnom programu kojemu je funkcija agregiranje sadržaja. Element vijesti može u sebi sadržavati i ostale opcionalne oznake koje služe za pružanje dodatnih informacija o samoj vijesti. Neki od tih oznaka predstavljaju datum objave vijesti, poveznicu na vijest, kratki opis vijesti, autora, kategoriju vijesti itd.

### 3.1.2. Visual Studio, .NET okvir te C# programski jezik

Visual Studio predstavlja razvojno okruženje kojeg razvija tvrtka Microsoft. Prva inačica izdana je 1997. godine. Od tada je razvojno okruženje prošlo kroz značajne preinake te je zadnja inačica izdana 2017. godine. Služi za razvoj raznih računalnih programa, web servisa, web aplikacija, web stranica, mobilnih aplikacija te računalnih igara. Podržava mnoštvo programskih jezika od kojih su najznačajniji C#, C++, JavaScript, XML, HTML i ostale pomoću određenih nadogradnji. Prema [24], podržava *IntelliSense* alat koji predlaže moguće završetke prethodno započetih naredbi ili prikazivanje metoda i sučelja koji se mogu koristiti u određenom projektu te ugrađene sustave za otklanjanje grešaka (engl. *debugger*) te dijagnostiku.

.NET okvir služi za razvoj računalnih programa i web servisa na Windows operacijskom sustavu. .NET Core je posebna inačica .NET-a pomoću koje je Microsoft omogućio razvoj programa, web servisa i ostaloga na drugim operacijskim sustavima – ponajviše na GNU/Linux te MacOS. Prema [25], sastoji se od dvije bitne komponente: posebne okoline u kojoj se pokreću programi (CLR) te od biblioteka klasa. Biblioteka klasa u sebi sadržava mnoštvo klasa kojima je glavni cilj pružiti svojevrsnu API funkcionalnost programerima. Primjerice programeru može biti na raspolaganju klasa za ostvarivanje veze prema bazi podataka iz .NET okvira. Zatim ta klasa može koristiti 5 drugih pomoćnih klasa iz okvira koje za cilj imaju ostvarivanje veze prema bazi podataka, rukovanju greškama i iznimkama te posebnom rukovanju podacima. Taj postupak pozadinskog odvijanja radnji programeru ne treba biti prioritet za pokušaj implementiranja sličnih funkcionalnosti jer je tu kompleksnost ostvarivanja veze prema bazi podataka razvio netko drugi te fokus programera treba biti na korištenju raspoloživih alata. C# je objektno orijentirani programski jezik koji se koristi za razvoj raznih programa i servisa unutar .NET okvira. Podržava nasljeđivanje, enkapsulaciju i polimorfizam. Nastao je kao konkurent Java programskom jeziku.

### 3.1.3. NEST

NEST predstavlja biblioteku koja se u okruženje raspoloživih biblioteka Visual Studija uključuje iz upravitelja paketa. Prema [26], podržava svu funkcionalnost koju podržava i Elasticsearch API. Glavna funkcija NEST biblioteke jest pružanje C# metoda kojima se opisuje povezivanje,

indeksiranje, preslikavanje dokumenata, pretraživanje i ostale funkcionalnosti koje se prilikom postupka prevođenja projekta pretvaraju u JSON oblik korišten za komunikaciju s API-jem Elasticsearch servisa. Također omogućava i otklanjanje grešaka te pregled koraka izvršavanja određenih upita na Elasticsearch servis. Postoje dva tipa NEST klijenta: klijent visoke razine i klijent niske razine.

### 3.1.4. Postman

Postman je alat koji je prvotno razvijen kao dodatak za web preglednik Google Chrome. Omogućavao je ispitivanje vlastitih web API-ja koristeći jednostavno korisničko sučelje. S obzirom na to da je popularnost drastično rasla, kompanija Postdot Technologies odučila je napraviti program koji je kompatibilan s najkorištenijim operacijskim sustavima. Program Postman ima mogućnosti kreiranja više kartica u kojoj svaka može imati svrhu neovisno o ostalim karticama. Omogućava odabir HTTP metoda, URI-ja, autorizaciju, definiciju sadržaja zaglavlja zahtjeva, sadržaj zaglavlja te definiranje testova pomoću kojih se jednim klikom miša može na predefiniran način izvršiti testiranje API-ja. Također omogućava i spremanje srodnih upita u kolekcije.

### 3.1.5. Elasticsearch

Elasticsearch ime je za sustav koji objedinjuje više samostalnih programskih komponenti koje će ukratko biti pojašnjene u nastavku te će se također dati pregled pojmova i funkcionalnosti Elasticsearch servisa korištenih za realizaciju rada. Sastoji se od Apache Lucene biblioteke, Kibane te Logstash programa. Apache Lucene predstavlja biblioteku otvorenog koda koja za cilj ima vršiti tekstualnu pretragu sadržaja u gotovo stvarnom vremenu. Pisana je Java programskim jezikom te pruža svoje mogućnosti pretrage preko raspoloživog API-ja kojeg koristi Elasticsearch. Prema [27], prva inačica biblioteke nastala je 1999. godine, a napisao ju je Doug Cutting. Ono što je bitno napomenuti za Apache Lucene jest da omogućava pretragu na bilo kojem tipu dokumenta koji sadrži tekstualna polja, tj. omogućava pretragu na PDF, word, HTML dokumentima. Najpoznatiji projekti koji su nastali neposrednom uporabom *Apache Lucene* biblioteke te koji proširuju njenu funkcionalnost su Apache Solr te Elasticsearch. To su međusobno dva glavna konkurenta u području tekstualnih pretraživača otvorenog koda. Oba projekta besplatno pružaju svoje osnovne funkcionalnosti široj javnosti te imaju i sustav koji je namijenjen korporacijama i centrima koji imaju veliki protok informacija.

Program *Kibana* ima za cilj vizualizirati stanje računala na kojem se udomljava Elasticsearch servis, stanje indeksa, stanje prometa prema indeksu itd. Pruža mogućnosti kreiranja stupčastih,

tortnih i sankijevih dijagrama. Također pruža mogućnosti ucrtavanja geografskih lokacija s kojih korisnici pristupaju servisu. Strojnim učenjem omogućava se uočavanje anomalija pri radu servisa kako bi se mogle poduzeti odgovarajuće mjere.

Program *Logstash* ima funkciju obilježavanja bitnih događaja pri radu sustava, komunikaciji s bazama podataka te načinom korištenja sa svrhom pružanja dodatnih informacija administratoru sustava.

Elasticsearch predstavlja distribuirani sustav pretrage. Prema [29], u distribuiranom sustavu Elasticsearcha postoje nakupine računala (engl. *cluster*) koja obavljaju sličnu funkcionalnost, bilo to puštanje aplikacije koja koristi servis u produkciju, razna testiranja ili spremanje informacija o mogućim greškama pri radu. U nakupini računala sudjeluju zasebna računala nazvana čvorovi (engl. *node)* koja međusobno dijele opseg opterećenja te povećavaju skalabilnost, performanse servisa i otpornost na kvarove. Svaki čvor u sebi može sadržavati jedan ili više indeksa. Indeks se u kontekstu prema relacijskim bazama podataka može promatrati kao baza podataka koja u sebi sprema tipove (tablice u kontekstu relacijskih baza podataka). Prije inačice Elasticsearch 6.0. dopuštalo se kreiranje više tipova po indeksu, no zbog složenosti indeksiranja dokumenata (u kontekstu relacijskih baza podataka spremanja podataka u relacijsku bazu) ta je podrška ukinuta. Također je ukinuta zbog načina na koji Apache Lucene vrši kompresiju dokumenata. Dokument u kontekstu prema relacijskim bazama podataka predstavlja jedan redak, n-torku. Dokumenti mogu sadržavati više polja (engl. *field*s) koja odgovaraju stupcima ili atributima relacijskih baza podataka. Najznačajniji tipovi podataka koje dokumenti mogu pohranjivati prema [28] su: *string*, *long*, *integer*, *short*, *double*, *float*, *date* te *geo-point*. Dokumenti se mogu definirati i indeksirati pomoću JSON formatiranog teksta koji se zatim usmjerava na Elasticsearch REST API. Također je moguće komunicirati s Elasticsearch servisom pomoću raspoloživih biblioteka za različite programske jezike od kojih su značajni Java, C# te PHP. Prema [29], u svrhu povećanja performansi pretraživanja, indeks se pri njegovom kreiranju može razdvojiti na manje dijelove zvane *krhotine* (engl. *shards*) ako je indeks prezahtjevan za datotečni sustav ili računalo na kojem se nalazi. U svrhu pružanja otpornosti na kvarove, tj. izrade sigurnosnih kopija, Elasticsearch omogućava kreiranje kopija krhotina indeksa. Ove navedene funkcionalnosti podržava svaki čvor unutar nakupine računala te Elasticsearch na transparentan način distribuira te krhotine i kopije nad kojima se također na udaljenim računalima može izvršavati pretraga kao da su locirani na jednom računalu što povećava skalabilnost i performanse pretrage. Za svrhu pohrane sadržaja vijesti (dokumenta) u indeks Elasticsearcha u .NET okruženju potrebno je definirati klasu koja sadrži svojstva (engl. *property*) koja će se zatim ugrađenim metodama NEST biblioteke preslikati

u predložak dokumenta kojeg će svaki unos podataka u indeks morati slijediti. Elasticsearch omogućava dva tipa preslikavanja: ručno (eksplicitno) i automatsko preslikavanje. Proces preslikavanja potrebno je odraditi prije indeksiranja prvog dokumenta ili pri kreiranju samoga indeksa što je preporučena praksa. Ručnim preslikavanjem može se odrediti tip podatka u koji će se preslikati svojstvo te koji tip analizatora će se primjenjivati na podatke koji će se unositi u indeks. Ručno preslikavanje preuzima prioritet neovisno o tome izvršava li se prije ili poslije automatskog preslikavanja. Analizator se unutar Elasticsearcha koristi kako bi tekstualni sadržaj prije procesa indeksiranja ili prije procesa pretrage sadržaja pretvorio u niz oznaka koje će se koristiti za pretragu. Analizatori uobičajeno sva slova postavljaju u mala slova te uklanjaju riječi poput veznika tj. riječi koji se često pojavljuju, a ne prenose informaciju. Neki od najkorištenijih tipova analizatora prema [30] su:

- Standardni – razdvaja riječi na njihovim granicama, uklanja interpunkcijske znakove, riječi koje upućuje u pretragu stavlja u mala slova i uklanja često pojavljivane riječi određenih jezika (engl. *stop words*),
- Jednostavni – razdvaja primljene riječi u izraze svaki put kada naiđe na simbol koji se razlikuje od slova te također pretvara izraze u zapis malim slovima,
- Analizator praznog mjesta – razdvaja riječi u izraze svaki puta kada naiđe na simbol praznog mjesta te ne pretvara izraze u mala slova, tj. održava ih onakvima kako su proslijeđeni,
- Ostali analizatori pojašnjeni su u [30].

U svrhu pretraživanja vijesti *Quicksearchom* koriste se dva tipa upita koje omogućava Elasticsearch: *match phrase* te *match* koji svaki zasebno ima svoje posebnosti pri pristupu pretraživanja sadržaja koji će ukratko biti opisani u nastavku.

*Match* – Prema [31], predstavlja *bool* tip upita. To znači da riječi koje servis primi za pretragu prvo prođu kroz analizator koji stvori izraze te se nakon toga vrši pretraga u ovisnosti o ILI ili I logičkoj funkciji nad tim izrazima. Podržava i dodane opcije poput premještanja slova te definiranje broja slova upita koja se smiju razlikovati od onih slova nad kojim se vrši pretraga. Zapravo predstavlja Levenshteinovu udaljenost između dvije riječi koja smije postojati da bi rezultat upita bio uspješan. Elasticsearch podražava postavljanje Levenshteinove udaljenosti na vrijednost od 0-2 u ovisnosti o broju slova u upitu.

*Match phrase* – Prema [32], na riječi ili skupu riječi koje dolaze kao upit na servis vrši se analiza nekom od gore navedenih analizatora. Zatim se vrši pretraga po točno onim izrazima i onim redoslijedom koje je analizator dao kao rezultat. Moguće je dodati opciju *slop* kojom se

omogućava transpozicija cijelih izraza, ali svi izrazi moraju biti prisutni nad poljem pretrage jer u protivnom sustav neće naći rezultat pretrage. Primjerice, da se u indeksu u nekom dokumentu nalaze sljedeće riječi „snažno stolno računalo“, a u upit se pošalju riječi „računalo snažno“ i da je vrijednost *slop* parametra 3, sustav bi pronašao odgovarajući dokument i vratio ga kao rezultat.

Za bilo koji tip upita prije njegova poziva moguće je specificirati tip analizatora koji će se koristi. U protivnom se odabire eksplicitno definiran analizator prilikom kreiranja indeksa ili se koristi podrazumijevani analizator prilikom iniciranja pretrage.

## 3.2. Način rada sustava i korištenje rješenja

Za uspješnu realizaciju problema pretrage vijesti pomoću *QuickSearcha* prvotno je bilo potrebno napraviti funkcionalnu razradu problema. Trebao se pronaći efikasan način pribavljanja vijesti s udaljenih izvora. To je ostvareno korištenjem *System.Xml.Linq* biblioteke. Ta biblioteka unutar sebe ima implementiranu metodu pomoću koje se u varijablu tipa *XElement* sprema protok podataka koji može biti definiran pomoću datoteke ili jedinstvenog URI-ja na koji se šalje GET zahtjev pomoću HTTP protokola te se tako pomoću odgovora dobiva sadržaj vijesti. Vijesti koje su korištene u ovom rješenju obuhvaćaju RSS 2.0 standard koji je formatiran XML-om. Najbitnije XML oznake u RSS dokumentu čine oznake naslova vijesti, poveznice na vijest te kratkog opisa sadržaja vijesti te metapodaci poput oznaka za jedinstveni identifikator vijesti te datum koji sadržava i vrijeme određene vremenske zone gdje se nalazi izvor objave cjelokupnog sadržaja vijesti. U nastavku na slici 3.1. prikazan je skraćeni sadržaj vijesti s navedenog URI-ja: https://www.phoronix.com/rss.php.

```
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
    <title>Phoronix</title>
    <link>https://www.phoronix.com/</link>
    <description>Linux Hardware Reviews &amp;amp; News</description>
    <language>en-us</language>
    <item>
      <title>AMD AOCC 1.2 Code Compiler Offers Some Performance Benefits For EPYC</title>
      <link>http://www.phoronix.com/vr.php?view=26316</link>
      <guid>http://www.phoronix.com/vr.php?view=26316</guid>
      <description>Last month AMD released the AOCC 1.2 compiler for Zen systems. This updated version of
their branched LLVM/Clang compiler with extra patches/optimizations for Zen CPUs was re-based to the
LLVM/Clang 6.0 code-base while also adding in experimental FLANG support for Fortran compilation and
various other unlisted changes to their "znver1" patch-set. Here's a look at how the performance compares with
AOCC 1.2 to LLVM Clang 6.0 and GCC 7/8 C/C++ compilers.</description>
      <pubDate>Sun, 20 May 2018 10:00:00 -0400</pubDate>
    </item>
```

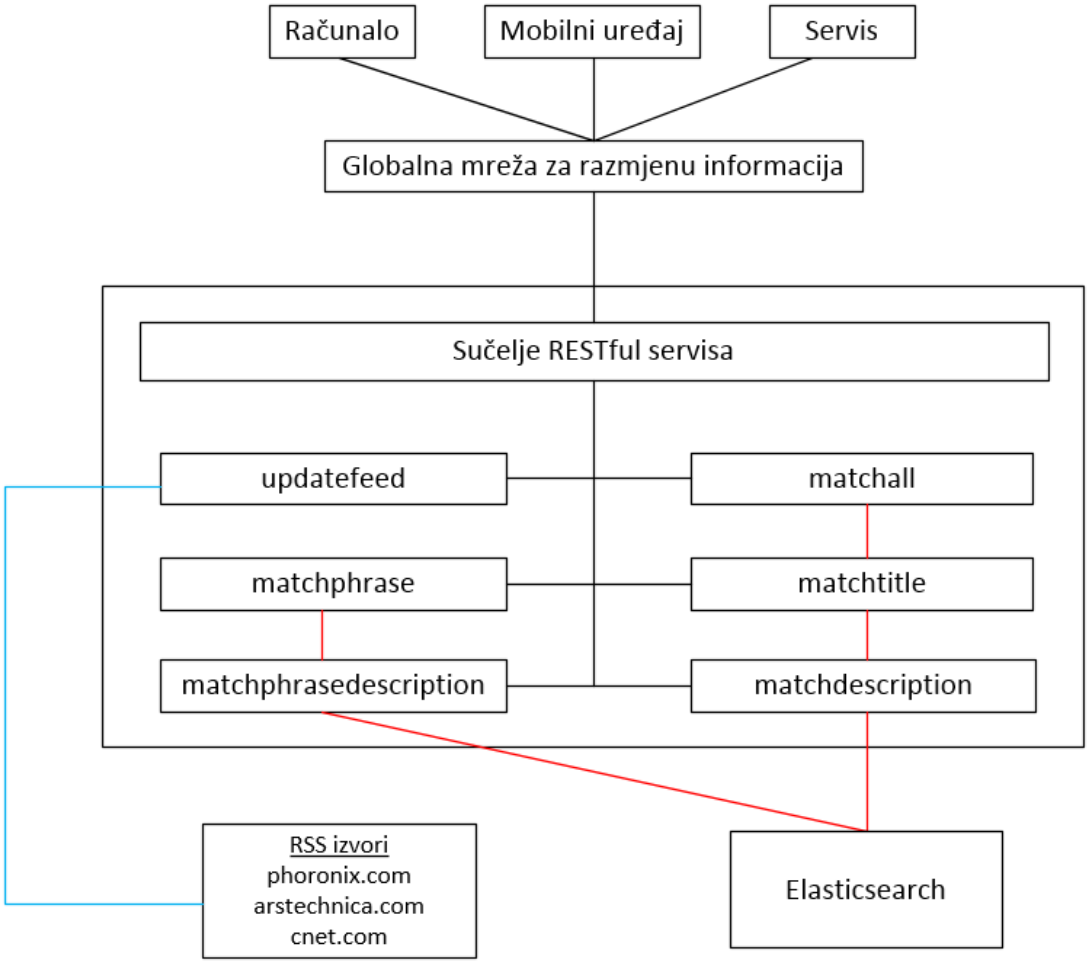
```
    <item>
      <title>Steam Controller Kernel Driver Is Landing In The Linux 4.18 Kernel</title>
      <link>http://www.phoronix.com/scan.php?page=news_item&amp;px=Steam-Controller-Linux-4.18</link>
      <guid>http://www.phoronix.com/scan.php?page=news_item&amp;px=Steam-Controller-Linux-
4.18</guid>
      <description>The Linux 4.18 kernel will feature the initial Steam Controller kernel driver that works
without having to use the Steam client or using third-party user-space applications like the SC-Controller
application...</description>
      <pubDate>Sun, 20 May 2018 07:05:29 -0400</pubDate>
    </item>
  </channel>
</rss>
```

Slika 3.1. *Sažeti prikaz vijesti*

U praksi u jednom dokumentu postoji barem deset vijesti, no za primjer iteracije po primljenim vijestima, prikazan je primjer sa samo dvije vijesti. Kao što je vidljivo na slici 3.1. postoji jedan korijenski element u XML dokumentu zvan rss s atributom *version* koji specificira u kojoj inačici RSS-a je realiziran sadržaj vijesti. Zatim neposredno ispod slijede informacije o sjedištu koje održava navedene informacije o vijestima. Bitno je napomenuti da je nužno da oznaka *guid* bude jedinstvena pa većina sjedišta u sadržaj te oznake stavljaju poveznicu (URI) na vijest pošto poveznica po definiciji također mora biti jedinstvena na cijeloj globalnoj mreži za razmjenu informacija.

Nakon što se u program uspješno unese sadržaj vijesti jednog od sjedišta, slijedi iteracija po XML dokumentu. Navedeno se izvršava automatski uporabom Elements("channel").Elements("item") metoda na prethodnu varijablu koja sadrži cijeli sadržaj vijesti. Navedeno se može ostvariti zbog toga što objekt tipa *XElement* podržava kreiranje „stabla“ po kojem se može kretati iterator koji će prikupljati potrebne informacije. U svrhu iteriranja koristi se *LINQ query,* sustav svojstven C# programskom jeziku, koji će u listu odabrati sadržaj vijesti, tj. naslov, poveznicu, identifikator, kratki opis i datum te će se u konstruktor vlastito definirane klase poslati takav sadržaj vijesti gdje će se izvršiti „filtriranje“ naslova, poveznice, datuma vijesti i kratkog sadržaja. Nužno je napomenuti da se za spremanje dokumenata vrši konverzija svih datuma i vremena vijesti od svakog sjedišta u UTC format zbog mogućih otkrivanja problema ili zbog statistike izračuna opterećenja web servisa kada bi se mogao očekivati najveći priljev vijesti ili prometa na tražilicu. Zatim se iterativno za svaki element tako generirane liste objekata poziva konstruktor klase koja služi za preslikavanje sadržaja te klase u shemu dokumenta koja služi za indeksiranje dokumenata u indeks Elasticsearcha. U toj klasi bilo je potrebno pomoću atributa specificirati NEST klijentu da jedinstveni identifikator dokumenta koristi MD5 sumu izračunatu iz sadržaja vijesti, tj. iz naslova, poveznice i datuma objave. Svrha takve radnje jest prepustiti sistemu Elasticsearcha

automatsko rukovanje mogućim duplikatima vijesti kada bi se pojavila potreba za ažuriranjem vijesti. Zbog toga što Elasticsearch nalaže potrebu da svaki novi dokument koji se pokuša indeksirati ima jedinstveni identifikator, u idealni slučaj dolazi MD5 suma za koju se pretpostavlja da će sadržaj koji će primati kao parametar ostati stalan, tj. da se sadržaj prethodnih vijesti prilikom njihova ažuriranja neće mijenjati te da se takav dokument neće spremiti kao duplikat u indeks zbog toga što dokument s jedinstvenim identifikatorom izračunatim pomoću MD5 sume već postoji u indeksu. U svrhu jednostavnijeg praćenja narednih odlomaka na slici 3.2. prikazane su komponente koje čine realizirano programsko rješenje *QuickSercha*.




Slika 3.2. *Skica ostvarenog sustava*

Za svrhu otvaranja mogućnosti ažuriranja sadržaja široj javnosti ili nekolicini administratora web servisa, u razvojnom okruženju Visual Studio kreiran je upravljač koji će rukovati akcijom ažuriranja sadržaja. Ujedno bilo kakvo kreiranje upravljača posljedično veže omogućavanje vanjskim korisnicima korištenje web servisa pomoću REST API-ja. U slučaju da se želi onemogućiti pristup određenim upravljačima korisnicima koji pripadaju u određene grupe, potrebno je dodati posebne atribute iznad definicije samog upravljača ili iznad definicija metoda koje se nalaze u upravljaču. U klasi upravljača potrebno je definirati metodu i atribut nad metodom

kojim će se naznačiti da metoda rukuje GET HTTP zahtjevom koji će za posljedicu izazvati ažuriranje sadržaja vijesti. Metoda sadržava tekstualno polje koje sadržava URI-je koji nude sadržaj svojih vijesti široj javnosti. Skalabilnost takvog pristupa je tome što administrator sam može dodati nove poveznice na druga sjedišta, te će sustav automatski izvršiti ažuriranje sjedišta vijesti ovisno o broju elemenata polja koje sadržava poveznice. Zatim se poziva petlja koja će proslijediti svaku poveznicu na dokument vijesti klasi koja je zadužena za filtriranje sadržaja vijesti te će se pozvati metoda koja je zadužena za spremanje filtriranog sadržaja vijesti u indeks. Metoda za spremanje kolekcije dokumenata kreira varijablu u koju će se spremiti rezultat odgovora Elasticsearch servisa te će se kontrola vratiti u metodu koja rukuje GET HTTP zahtjevom. Ondje se u ovisnosti o uspješnosti indeksiranja kreira objekt pomoću kojeg će se kreirati generički odgovor na izvršenu radnju. Objekt će biti automatski generiran u JSON formatu zbog podešavanja postavke web servisa koja mijenja podrazumijevano ponašanje servisa da odgovara na zahtjeve samo u JSON formatu koji je više u duhu REST arhitekturalnog stila zbog svoje jednostavnosti i razumljivosti.

Zatim slijedi kreiranje pet ostalih upravljača kojima je svrha pretraživanje sadržaja pohranjenih u indeks. Četiri od pet upravljača za ispravno funkcioniranje moraju imati prisutnu varijablu za pretraživanje te njenu „vrijednost“, tj. riječ od koje će se tražiti željeni naslov vijesti ili kratki opis vijesti pohranjenih unutar dokumenata u indeksu. Preciznije rečeno, nakon unosa jedinstvenog URI-ja u adresnu traku web preglednika, nužno je dodati par „naziv varijable – vrijednost varijable“ u dio upita URI-ja. Poslije imena upravljača u adresnoj traci potrebno je dodati znak '?' koji označava početak parova „naziv varijable – vrijednost varijable“ koji će se proslijediti sustavu za rukovanje parametrima te će taj sustav nad njima izvršiti automatsko određivanje tipa vrijednosti koje sadržavaju te će se kao takvi proslijediti u određeni parametar metode koja se nalazi u upravljaču. Kao opcionalni parametar upravljaču se može proslijediti varijabla s brojem koja će se zatim iz upravljača proslijediti metodi koja vrši pretragu sadržaja gdje će služiti za ograničenje broja najbitnijih rezultata u slučaju ako se to želi ostvariti pošto sustav Elasticsearcha pri vrhu stavlja rezultate koje smatra najbitnijima. Sustav Elasticsearcha određuje najbitnije rezultate na temelju ugrađenog algoritma (TF/IDF) prema [33] koji određuje ostvarene „bodove“. Točnije, ima ugrađeni sustav „bodovanja“ prema učestalosti pojavljivanja tražene riječi unutar određenog polja nad kojim se vrši pretraga, učestalosti pojavljivanja tražene riječi u indeksu te duljini polja u kojemu se pojavljuje tražena riječ. Što je polje unutar kojega se nalazi tražena riječ dulje to je u rezultatu dodijeljeno manje „bodova“ za to polje. Vrijedi i obrat. U slučaju da postoji više parametara na jedan URI, nužno ih je odvojiti znakom '&'. Čak i ako se zamijeni redoslijed

parametara poslanih upravljaču od onoga kako je definirano u metodi koju se poziva da rukuje zahtjevom, sustav će sam znati proslijediti parametre ako se razlikuju po tipu vrijednosti koju pohranjuju.

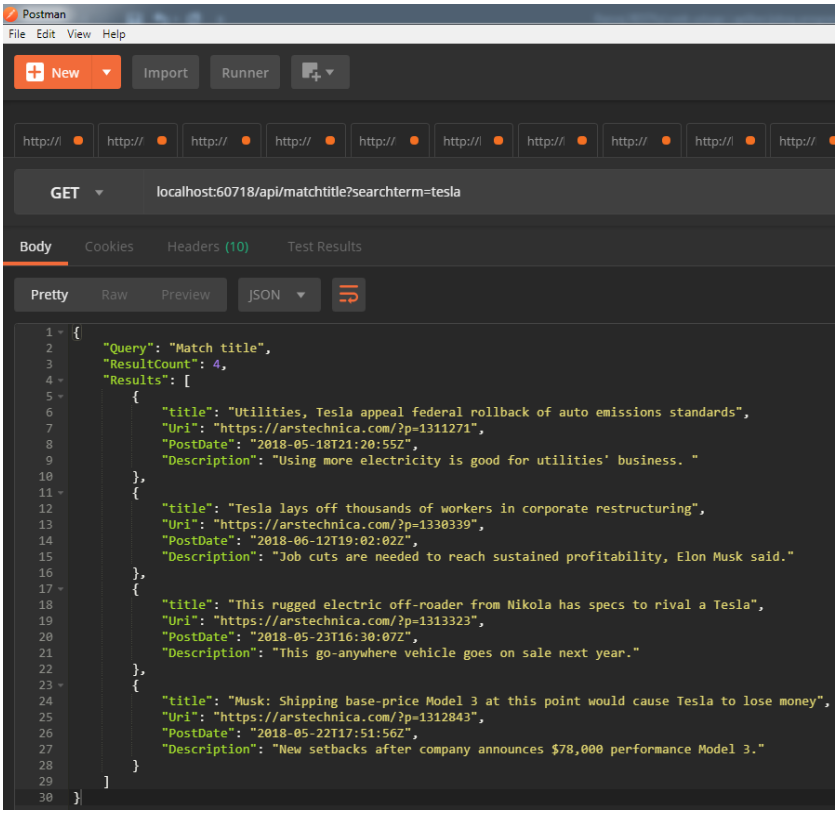
U daljnjem izlaganju bit će prezentirana tri od pet upravljača. Razlog te odluke jest u tome što upravljači za pretragu kratkog sadržaja vijesti koriste iste metode pretrage kao i dva upravljača koja pretražuju naslove vijesti i što se razlikuju po tome kako upravljači prezentiraju sadržaj kao odgovor na zahtjev u ovisnosti koji od upita su prihvatili – upita na naslov vijesti ili upita na opis sadržaja. Detaljnije će biti prikazano u potpoglavlju 3.3. gdje će biti prikazani rezultati poziva svih upravljača.

Prvi od upravljača služi za posluživanje broja rezultata neovisno o terminu za pretraživanje sadržaja. Definicija metode upravljača omogućava provjeru jedinog opcionalnog parametra kojeg može primiti – broj rezultata koji će se vratiti iz Elasticsearch servisa. Ograničenje vezano za najveći mogući broj rezultata koji se mogu dobiti kao odgovor, definirano je u posebnoj klasi. Svrha te klase jest raspoloživost centraliziranog mjesta za promjenu najvećeg, najmanjeg i podrazumijevanog broja rezultata koji se mogu usmjeriti u pretragu. Ograničenje najvećeg i najmanjeg broja rezultat uneseno je u sustav da se korisniku onemogući preveliki broj rezultata te unos negativnih brojeva. Zbog prirode upita koji se mogu usmjeravati na Elasticsearch servis nužno je pružiti tu vrstu kontrole korisniku API-ja. Također, metoda nakon provjere primljenog parametra veličine prosljeđuje taj isti parametar vlastitoj metodi. U toj metodi definirana je varijabla u koju će se iz rezultata upita Elasticsearch servisa filtrirati samo onaj dio rezultata koji sadrži dokumente koje je servis uspio pronaći. Također se ista radnja ponavlja i kod naredna dva upravljača. Zatim se tijek izvođenja vraća u metodu upravljača gdje se pomoću klasa *Newtonsoft.JSON* biblioteke kreira JSON objekt u kojem se formatira rezultat zadanog upita. Taj objekt sadržava informacije o tipu upita koji je primljen, odabranom broju rezultata pomoću parametra, broj rezultata kojeg je uspio vratiti Elasticserach servis, te sami rezultati, tj. dokumenti koji su se vratili iz servisa. Zatim se nad dokumentima u JSON objektu izvršava *LINQ* dio koda gdje se dinamički u ovisnosti o količini dokumenata koji su dohvaćeni, kreiraju novi JSON objekti koji će biti prikazani zajedno s ostalim parametrima JSON objekta kao rezultat poslanog upita u web pregledniku.

Drugi od upravljača za funkciju ima poslužiti *matchphrase* upit na Elasticsearch servis, a poziva istoimenu metodu. Za uspješno prepoznavanje upravljača od strane sustava usmjeravanja upita, nužno je pri pozivu upravljača proslijediti varijablu u kojoj je spremljena vrijednost izraza kojeg se želi pronaći u indeksu, tj. varijablu *searchterm* čija će uporaba biti prikazana u narednom

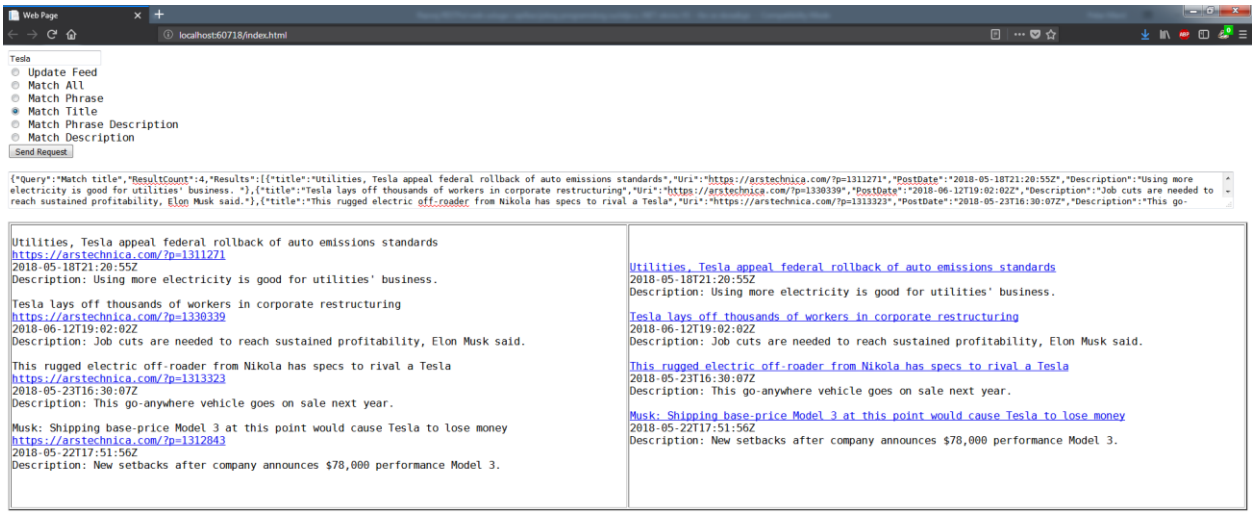
potpoglavlju. Također se kao dodatni parametar upravljaču može proslijediti broj kojim se specificira najveći broj rezultata koji će se moći prikazati kao rezultat upita. Ako se upravljaču ne pošalje varijabla koja sadržava upit, sustav za usmjeravanje upita neće moći pronaći adekvatan upravljač te će generirati poruku u kojoj navodi da ne može prepoznati upravljač. Ista situacija dogodila bi se i ako bi se upravljaču poslao samo broj rezultata. Isto ponašanje imaju i preostala tri upravljača, tj. jedan za pretragu vijesti, ali drugom metodom te dva koji pretražuju polje kratkog sadržaja vijesti. Kada se uspješno proslijede parametri metodi iz upravljača, poziva se vlastita metoda kojoj se prosljeđuje varijabla koja sadrži riječ pretrage te podrazumijevani ili definirani broj rezultata. Nakon što se kontrola preda NEST metodi, u nju se uvrštavaju parametri poslani iz metode upravljača te se modificiraju vrijednosti koje će pomoći povećati relevantnost samih rezultata ovisno u danom upitu. Također se nakon završetka pretrage u Elasticsearch servisu rezultati vraćaju nazad u metodu upravljača te se opet dinamički vrši generiranje JSON objekta koji će služiti kao odgovor na navedenu radnju upravljača.

Treći od upravljača poslužuje *match* upite na Elasticsearch servis. Funkcionalnost *match* naredbe odrađena je u odjeljku 3.1.5. kao i dodatni parametri koji pomažu u pronalasku rezultata. Tom upravljaču također je potrebno proslijediti varijablu u koju će se spremiti vrijednost traženog upita. Ako ne proslijedi vrijednost, sustav će pojaviti grešku. Isto bi se dogodilo da se proslijedi samo varijabla za broj rezultata. Zatim se ispituje je li odabrana adekvatna veličina rezultata te se kontrola predaje metodi koja poziva Elasticsearch servis te prosljeđuju parametri za pretraživanje i količinu rezultata. Nakon povratka rezultata također se filtriraju samo dokumenti od odgovora servisa. Zatim se kontrola vraća u upravljač gdje se kreira JSON objekt s formatiranjem potrebnim da opiše radnje tog upravljača. Tu se opet dinamički u ovisnosti broja rezultata pomoću *LINQ* koda generiraju objekti. Nakon toga upravljač vraća cjelokupni JSON objekt koji se pokazuje u pregledniku na HTTP odgovor koji se zatim može koristiti u svrhe razvoja vlastite aplikacije ili API-ja koji je ovisan o opisanom API-ju. Na slici 3.3. može se vidjeti korištenje Postman alata za upućivanje zahtjeva prema kreiranom API-ju korištenjem putanje URI-ja koja poziva *matchtitle* upravljač, riječi za pretraživanje u upitu te odgovora samog servisa na formatiran zahtjev.




Slika 3.3. *Prikaz sučelja Postman alata pri kreiranju upita*

Također je u svrhu demonstriranja rada web API-ja kreirana i jednostavna hipertekstualna web stranica. Takva web stranica služi kao prototip kako bi se API mogao koristiti za neku ozbiljniju primjenu kao primjerice za forume, blogove, vijesti, oglasnike itd. Na stranici se nalazi šest kontrola i jedna tipka za slanje zahtjeva. Svaka kontrola odgovara jednom upravljaču te se u ovisnosti o odabranoj kontroli omogućava ili onemogućava unos teksta pomoću kojega će se tražiti sadržaj preko web servisa. Pritiskom na svaku kontrolu, poziva se odgovarajuća funkcija pomoću koje se vrši instanciranje varijabli u koje se stavlja određeni tekst kojim se definira potpuni izgled URI-ja. Pomoću takvih varijabli poziva se dodatna funkcija gdje se te varijable prenose kao parametri. Zatim se vrši definiranje baznog URI-ja na kojeg će se vezati sadržaj prethodno definirane varijable. Nakon toga slijedi definiranje jednostavne AJAX (engl. *Asynchronous JavaScript And XML*) funkcije, tj. XMLHttpRequest. Pomoću te funkcije omogućava se prijevod odgovora web servisa u JavaScript objekt te se zatim dinamički generira HTML kod koji se pokazuje u tablici vidljivoj na slici 3.4. Lijevi stupac tablice definiran je da sva svojstva rezultata pokaže zasebno dok je u desnom stupcu definirano da naslov vijesti označi kao hipervezu na sjedište gdje je vijest objavljena. Također se postavlja i objekt *textarea* na nemanipulirani odgovor web servisa kako bi se bolje prikazalo što definira kod u pozadini. Na slici 3.4. prikazan je primjer korištenja kreirane web stranice u svrhu interakcije s web servisom.




Slika 3.4. *Prikaz kreirane web stranice*

## 3.3. Testiranje rješenja

U ovome potpoglavlju bit će detaljno prikazan način na koji funkcionira kreiran API *Quicksearcha*. Prikazat će se dozvoljene i nedozvoljene radnje na svakom upravljaču zasebno. Također će biti izloženi komentari i obrazloženja neposredno ispod upravljača. Zbog fizičkih ograničenja papira nemoguće je prikazati funkcioniranje API-ja preko web preglednika jer bi tekst na slikama bio nečitljiv, a slika bi zauzimala mnogo prostora na papiru. Stoga je odabrana metoda gdje će se pomoću tablica prikazati nazivi upravljača, zahtjev u obliku URI-ja koji se usmjerava na API kreiranog web servisa te prikladni odgovori.

Tablica 3.1. *Prikaz upravljača updatefeed*

| **Upravljač** | ***updatefeed*** |
|---|---|
| **URI** | http://localhost:60718/api/updatefeed |
| **Odgovor servisa** | [<br>{<br>"HttpMethod": "POST",<br>"HttpStatusCode": "200",<br>"ResponseMimeType": "application/json; charset=UTF-8",<br>"Success": true,<br>"Uri": "http://localhost:9200/visual_studio/_bulk?pretty=true",<br>"FeedUri": "http://feeds.arstechnica.com/arstechnica/index/"<br>},<br>{<br>"HttpMethod": "POST",<br>"HttpStatusCode": "200",<br>"ResponseMimeType": "application/json; charset=UTF-8",<br>"Success": true,<br>"Uri": "http://localhost:9200/visual_studio/_bulk?pretty=true",<br>"FeedUri": "https://www.phoronix.com/rss.php"<br>}, |

```
    {
      "HttpMethod": "POST",
      "HttpStatusCode": "200",
      "ResponseMimeType": "application/json; charset=UTF-8",
      "Success": true,
      "Uri": "http://localhost:9200/visual_studio/_bulk?pretty=true",
      "FeedUri": "http://feed.cnet.com/feed/topics/tech-industry"
    },
    {
      "HttpMethod": "POST",
      "HttpStatusCode": "200",
      "ResponseMimeType": "application/json; charset=UTF-8",
      "Success": true,
      "Uri": "http://localhost:9200/visual_studio/_bulk?pretty=true",
      "FeedUri": "http://feeds.feedburner.com/HowToGeek"
    },
    {
      "HttpMethod": "POST",
      "HttpStatusCode": "200",
      "ResponseMimeType": "application/json; charset=UTF-8",
      "Success": true,
      "Uri": "http://localhost:9200/visual_studio/_bulk?pretty=true",
      "FeedUri": "http://feeds.nationalgeographic.com/ng/News/News_Main"
    }
]
```


Ovaj odgovor web servisa služi za generičko prikazivanje onih poveznica na vijesti koje su uspjele biti ažurirane, a koje se nalaze definirane u upravljaču zaduženom za ažuriranje sadržaja.

Tablica 3.2. *Prikaz upravljača matchall*

| **Upravljač** | ***matchall*** |
|---|---|
| **URI** | http://localhost:60718/api/matchall?size=200 |
| **Parametar upita** | size 1-500 |
| **Odgovor servisa** | {<br>"Query": "Match all",<br>"SelectedSize": 200,<br>"ResultCount": 200,<br>"Results": [<br>{<br>"title": "KDE Plasma 5.13 Beta Released With A Compelling Number Of Improvements",<br>"Uri": "http://www.phoronix.com/scan.php?page=news_item&px=KDE-Plasma-5.13-Beta",<br>"PostDate": "2018-05-18T16:18:54Z",<br>"Description": "KDE Plasma 5.13 is out in beta form today ahead of its planned release in June. Plasma 5.13 has ended up being a very compelling and huge upgrade for the KDE Plasma 5 desktop..."<br>},<br>{<br>"title": "Ubuntu 18.10 Aims To Lower Power Use, Default To New Desktop Theme",<br>"Uri": "http://www.phoronix.com/scan.php?page=news_item&px=Ubuntu-18.10-Desktop-Plans",<br>"PostDate": "2018-05-18T15:53:24Z", |

"Description": "Will Cooke, the Director of the Ubuntu Desktop at Canonical, has outlined the major desktop plans for the Ubuntu 18.10 \"Cosmic Cuttlefish\" cycle..."
},  …

Ovaj upravljač zadužen je za prikazivanje svih vijesti neovisno o sadržaju naslova ili kratkog sadržaja u ovisnosti o podrazumijevanom ili opcionalnom parametru gdje se definira broj rezultata. Taj broj moguće je promijeniti u posebnoj klasi koja sadržava podrazumijevane i najveće dopuštene veličine koje se mogu koristiti za broj pribavljanja rezultata pretrage na indeksu Elasticsearcha.

Tablica 3.3. *Prikaz upravljača matchphrase*

| **Upravljač** | ***matchphrase*** | |
|---|---|---|
| **URI** | http://localhost:60718/api/matchphrase?searchterm=microsoft&size=2 | |
| **Parametri upita** | searchterm | *riječ ili riječi za pretragu* |
| | size | 1-50 |
| **Odgovor servisa** | {<br>"Query": "Match phrase",<br>"ResultCount": 2,<br>"Results": [<br>{<br>"title": "How to Convert a Microsoft Word Document to a PDF",<br>"Uri": "https://www.howtogeek.com/352668/how-to-convert-a-microsoft-word-document-to-a-pdf/",<br>"PostDate": "2018-05-21T14:24:00Z",<br>"Description": "PDFs are handy for distributing documents so that they’re seen the same way by all parties. Typically, you’ll create documents using another app, and then convert them to PDF. Here’s how to do it for a Microsoft Word document.<img src=\"http://feeds.feedburner.com/~r/HowToGeek/~4/nOsImGnkdL8\" height=\"1\" width=\"1\" alt=\"\"/>"<br>},<br>{<br>"title": "How to Create, Edit, and View Microsoft Word Documents for Free",<br>"Uri": "https://www.howtogeek.com/351601/how-to-create-edit-and-view-microsoft-word-documents-for-free/",<br>"PostDate": "2018-05-21T10:40:00Z",<br>"Description": "There was a time when you had to have Microsoft Office installed to create, edit, or even view a Microsoft Word document. Thankfully, that’s no longer the case. There are a number of free alternatives for working with those Word documents people occasionally send you.<img src=\"http://feeds.feedburner.com/~r/HowToGeek/~4/HcqK-xfdqL0\" height=\"1\" width=\"1\" alt=\"\"/>"<br>}<br>]<br>} | |

Za upit matchphrase nužno je napomenuti da se pretraga vrši po točno onoj riječi ili riječima koji su poslani u upravljač. Drugačije rečeno, da se u upit poslala riječ „Microso“ sustav ne bi našao niti jednu vijest čiji bi naslov sadržavao tu riječ. No, u sljedećoj vrsti upravljača slanje riječi

„Microso“ u sustav za pretragu daje iste rezultate kao što *matchphrase* daje rezultate za riječ „Microsoft“.

Tablica 3.4. *Prikaz upravljača matchtitle*

| **Upravljač** | ***matchtitle*** |
|---|---|
| **URI** | http://localhost:60718/api/matchtitle?searchterm=microso&size=2 |
| **Parametri upita** | searchterm — *riječ ili riječi za pretragu*<br>size — 1-50 |
| **Odgovor servisa** | { "Query": "Match title", "ResultCount": 2, "Results": [ { "title": "How to Convert a Microsoft Word Document to a PDF", "Uri": "https://www.howtogeek.com/352668/how-to-convert-a-microsoft-word-document-to-a-pdf/", "PostDate": "2018-05-21T14:24:00Z", "Description": "PDFs are handy for distributing documents so that they’re seen the same way by all parties. Typically, you’ll create documents using another app, and then convert them to PDF. Here’s how to do it for a Microsoft Word document.<img src=\"http://feeds.feedburner.com/~r/HowToGeek/~4/nOsImGnkdL8\" height=\"1\" width=\"1\" alt=\"\"/>" }, … |

Kao što je vidljivo sustav interno vrši usporedbu unesenih riječi s onima koje se nalaze u naslovima vijesti. Razlog takvog ponašanja opisan je u odjeljku 3.1.5. gdje se pojašnjavaju sposobnosti funkcija koje se koriste za pretragu.

U nastavku će biti izložena preostala dva upravljača koji imaju svrhu proslijediti upit na Elasticsearch servis i to samo za polje kratkog opisa vijesti (engl. *description*). Pozivajući bilo koji od ta dva upravljača kreirani web API formatira polje kratkog opisa vijesti tako da stavi to polje kao prvo polje unutar JSON objekta, tj. unutar polja rezultata, kao što će biti vidljivo.

Tablica 3.5. *Prikaz upravljača matchphrasedescription*

| **Upravljač** | ***matchphrasedescription*** |
|---|---|
| **URI** | http://localhost:60718/api/matchphrasedescription?searchterm=motor&size=2 |
| **Parametri upita** | searchterm — *riječ ili riječi za pretragu*<br>size — 1-50 |
| **Odgovor servisa** | { "Query": "Match phrase description", "ResultCount": 1, "Results": [ { "Description": "\"Cost of normal dual motor AWD option is $5k,\" Tesla CEO says.", "Uri": "https://arstechnica.com/?p=1311773", "title": "Elon Musk tweet-announces a $78,000 performance Model 3 with all-wheel drive", |

```
            "PostDate": "2018-05-20T16:55:38Z"
        }
    ]
}
```


Kao što je vidljivo iz prikazanog primjera pozvan je odgovarajući upravljač s pripadnim parametrima za pretraživanje i brojem koji predstavlja koliko bi se rezultata trebalo prikazati kao odgovor na dani upit. Također je vidljivo da je u URI-ju proslijeđena količina rezultata definirana brojem 2, no sustav Elasticsearcha nije mogao pronaći ostale sadržaje vijesti u kojima bi se pojavila riječ „motor“. Stoga je upravljač formatirao ispis tako što je ključ ResultCount postavio na 1. U slučaju da je Elasticsearch pronašao 3 ili više kratkih sadržaja vijesti u kojima se pojavljuje riječ „motor“ iz upravljača bi se proslijedila samo dva najrelevantnija kratka sadržaja vijesti.

Tablica 3.6. *Prikaz upravljača matchdescription*

| **Upravljač** | ***matchdescription*** | |
|---|---|---|
| **URI** | http://localhost:60718/api/matchdescription?searchterm=nrmal mtr&size=2 | |
| **Parametri upita** | searchterm | *riječ ili riječi za pretragu* |
| | size | 1-50 |
| **Odgovor servisa** | {<br>"Query": "Match description",<br>"ResultCount": 1,<br>"Results": [<br>{<br>"Description": "\"Cost of normal dual motor AWD option is $5k,\" Tesla CEO says.",<br>"Uri": "https://arstechnica.com/?p=1311773",<br>"title": "Elon Musk tweet-announces a $78,000 performance Model 3 with all-wheel drive",<br>"PostDate": "2018-05-20T16:55:38Z"<br>}<br>]<br>} | |

S ovime završava kratki pregled funkcionalnosti kreiranog API-ja i korištenih upravljača te će se u zaključku izložiti moguća područja dorade rada te najvažnije odrednice iz teorijskog dijela rada.

# 4. ZAKLJUČAK

U ovome radu izloženi su standardi, protokoli i arhitekture koji omogućuju postojanje globalne mreže za razmjenu informacija. Pridržavanjem određenih pravila i načela kreiranja web aplikacija ili usluga može se pojednostaviti otkrivanje i korištenje njihovog aplikacijskog programskog sučelja koje se može koristiti za definiranje složenijih aplikacijskih programskih sučelja. Programsko rješenje za pretragu vijesti *QuickSearch* izrađeno je s ciljem centraliziranja vijesti različitih sjedišta te mogućnost pretraživanja koristeći kreirano aplikacijsko sučelje. *QuickSearch* za pretragu koristi REST završne točke Elasticsearch servisa pohranjene unutar NEST biblioteke te Microsoftov web api. Elasticsearch pruža pretragu nad dokumentima pohranjenim u indeksu u gotovo stvarnom vremenu, a Microsoftov web api omogućava definiranje potrebnih upravljača preko kojih će se vršiti upiti za pretragu vijesti. Testiranjem kreiranog aplikacijskog programskog sučelja nisu otkrivene nepravilnosti rada određenih upravljača te je također testiranjem pokazano da se za određene upite koristi transpozicija riječi pretrage kako bi se pronašle relevantne vijesti. Nedostaci kreiranog programskog rješenja jesu nemogućnost sjedinjenja različitih formata mrežnih vijesti te nemogućnost kreiranja odgovora u XML formatu. Unaprjeđenja rada mogu biti na pružanju pohrane i pretrage različitih formata vijesti, kreiranje sigurnosnih kopija, kreiranje posebnog sustava koji prati ponašanje kreiranog servisa ili pružanje pregovaranja o formatu sadržaja poruke koja će se slati kao odgovor na određeni zahtjev. Mogući radovi koji kao osnovu mogu uzeti ovaj rad jesu usporedba Elasticsearch i Apache Solr sustava za pretraživanje te usporedba vremena potrebnog za pretragu sadržaja u ovisnosti o broju dokumenata sadržanih u indeksu.

# LITERATURA


[1] Internet World Stats, „*World Internet Users Statistics*“, https://www.internetworldstats.com/stats.htm, pristup ostvaren 16.5.2018.

[2] J. Yannakopoulos, „*HyperText Transfer Protocol: A Short Course*“, http://condor.depaul.edu/dmumaugh/readings/handouts/SE435/HTTP/http.pdf, pristup ostvaren 1.6.2018.

[3] IETF, „*RFC 2396*“, https://www.ietf.org/rfc/rfc2396.txt, pristup ostvaren 1.6.2018.

[4] Robert F. MacInnis, „*Web Services*“, https://www.activeaether.com/documents/ActiveAether_Web-Services-Extended_09-28-17.pdf, pristup ostvaren 1.6.2018.

[5] W3Schools „*XML WSDL*“, https://www.w3schools.com/xml/xml_wsdl.asp, pristup ostvaren 31.5.2018.

[6] Mike P. Papazoglou, „*SOAP: Simple Object Access Protocol*“, https://www.cs.colorado.edu/~kena/classes/7818/f08/lectures/lecture_3_soap.pdf, pristup ostvaren 31.5.2018.

[7] W3C, „*SOAP version 1.2*“ , https://www.w3.org/TR/soap12-part0/#intro, pristup ostvaren 31.5.2018.

[8] UpWork, „*Soap vs REST*“, https://www.upwork.com/hiring/development/soap-vs-rest-comparing-two-apis/ , pristup ostvaren 31.5.2018.

[9] S. Mumbaikar, P.Padiya, „*Web Services Based On SOAP and REST Principles*“, International Journal of Scientific and Research Publications, Volume 3, 2013.

[10] R. T. Fielding, R. N. Taylor, J. R. Erenkrantz , M. M. Gorlick, J. Whitehead, R. Khare, and P. Oreizy, *„Reflections on the REST Architectural Style and Principled Design of the ModernWeb Architecture“,* Paderborn, Germany, 2017.

[11] Fielding, Roy Thomas, „*Architectural Styles and the Design of Network-based Software Architectures“*, Doctoral dissertation, University of California, Irvine, 2000.

[12] Tutorials Point, „*Software Architecture and Design Dana Flow Architecture*“, https://www.tutorialspoint.com/software_architecture_design/data_flow_architecture.htm, pristup ostvaren 29.5.2018.

[13] R.Hurbans, „*The Rest Architectural Styles“,* https://github.com/rishal-hurbans/The-REST-Architectural-Style*,* pristup ostvaren 30.5.2018.

[14] W3C, „*Extensible Markup Language (XML) 1.0 (Fifth Edition)*“, https://www.w3.org/TR/xml/, pristup ostvaren 26.5.2018.

[15] IETF*, „The JavaScript Object Notation (JSON) Data Interchange Format“,* https://tools.ietf.org/html/rfc7159*,* pristup ostvaren 26.5.2018.

[16] Todd Fredrich, „*RESTful Service Best Practices***“,** Pearson eCollege, 2013.

[17] Rainer Stropek, „*RESTful Web API Design*“, https://speakerdeck.com/rstropek/restful-web-api-design, pristup ostvaren 29.5.2018.

[18] Mark Massé*, „REST API Design Rulebook*“, SAD, 2011.

[19] Jamie Kurtz, „*ASP.NET MVC 4 and the Web API*“, SAD, 2013.

[20] Microsoft, „*Overview of ASP.NET Core MVC*“, https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-2.0, pristup ostvaren 27.5.2018.

[21] GeeksforGeeks, „*MVC Design Pattern*“, https://www.geeksforgeeks.org/mvc-design-pattern/, pristup ostvaren 28.5.2018.
[22] A. King , „*The Evolution of RSS*“, http://webreference.com/authoring/languages/xml/rss/1/3.html, pristup ostvaren 23.5.2018.
[23] W3C, „*Rss 2.0 specification*“, https://validator.w3.org/feed/docs/rss2.html, pristup ostvaren 23.5.2018.
[24] Microsoft, „*.NET development; Visual Studio*“, https://www.visualstudio.com/vs/features/net-development/ , pristup ostvaren 24.5.2018.
[25] Microsoft, „*Get started with the .NET Framework*“, https://docs.microsoft.com/en-us/dotnet/framework/get-started/ , pristup ostvaren 24.5.2018.
[26] Elastic.co, „*NEST – High level client*“, https://www.elastic.co/guide/en/elasticsearch/client/net-api/current/nest.html, pristup ostvaren 24.5.2018.
[27] G. Ingersoll, „*Better Search with Apache Lucene and Solr*“, https://web.archive.org/web/20120131154001/http://trijug.org/downloads/TriJug-11-07.pdf, pristup ostvaren 24.5.2018.
[28] Elastic.co, „*Field datatypes*“, https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-types.html, pristup ostvaren 25.5.2018.
[29] Elastic.co, „*Basic Concepts*“, https://www.elastic.co/guide/en/elasticsearch/reference/current/_basic_concepts.html, pristup ostvaren 25.5.2018.
[30] Elastic.co, „*Analyzers*“, https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html, pristup ostvaren 25.5.2018.
[31] Elastic.co, „*Match Query*“, https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-match-query.html, pristup ostvaren 25.5.2018.
[32] Elastic.co, „*Match Phrase Query*“, https://www.elastic.co/guide/en/elasticsearch/reference/6.2/query-dsl-match-query-phrase.html, pristup ostvaren 25.5.2018.
[33] Elastic.co „*Theory behind relevance scoring*“, https://www.elastic.co/guide/en/elasticsearch/guide/current/scoring-theory.html, pristup ostvaren 25.5.2018.
[34] Tiptopsecurity, „*How does HTTPS work?*“, https://tiptopsecurity.com/how-does-https-work-ssl-tls-explained/“, pristup ostvaren 12.6.2018.
[35] IETF, „*RFC6838*“, https://tools.ietf.org/html/rfc6838, pristup ostvaren 13.6.2018.
[36] IANA, „*Media Types*“, https://www.iana.org/assignments/media-types/media-types.xhtml, pristup ostvaren 13.6.2018.
[37] W3C, „*SOAP*“, https://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383526, pristup ostvaren 14.6.2018.
[38] IEEE, „*Roy T. Fielding: Understanding the REST Style*“, https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7122189, pristup ostvaren 15.6.2018.

[39] Wikimedia, „*WebService*“, https://commons.wikimedia.org/wiki/File:Webservice.svg, pristup ostvaren 15.6.2018.

# SAŽETAK

U ovome radu daje se kratki pregled HTTP protokola i URI standarda koji su temelji globalne mreže za razmjenu informacija kao i dva stila arhitekture za razvoj web servisa: SOAP i REST arhitekture s fokusom na REST arhitekturu. Pojašnjeni su glavni elementi koji čine SOAP arhitekturu – WSDL, UDDI te SOAP protokol za razmjenu poruka između distribuiranih sustava. S obzirom na složenost razvoja SOAP web servisa pojavila se REST arhitektura. U ovome radu pojašnjeni su osnovni stilovi arhitekture iz kojih je nastao REST kao njihov hibrid. Također su detaljno predstavljeni njezini elementi podataka, elementi konektora i komponente. Pridržavanjem opisanih šest ograničenja nametnutih REST-om web servisi postaju RESTful web servisi. Za prijenos informacija između web servisa koriste se XML i JSON prijenosni formati, za razvoj programa MVC oblikovni obrazac. U praktičnom dijelu rada detaljno su opisani alati i tehnologije potrebni za prikupljanje i pohranu vijesti s raznih web sjedišta sa svrhom omogućavanja pretrage sadržaja vijesti. Za pohranu i pretraživanje sadržaja vijesti korišten je program Elasticsearch, a za kreiranje aplikacijskog programskog sučelja preko kojeg se pristupa Elasticsearch-u te nužni upravljači za rukovanje zahtjevima usmjerenih prema web servisu kreirani su unutar .NET okruženja.

**Ključne riječi:** .NET, Elasticsearch, NEST, REST, SOAP, web usluga.

## ABSTRACT

### Developing a RESTful web service and an application programming interface using the .NET framework

In this paper, the short overview of HTTP protocol and URI standard as the foundation of World Wide Web is given as well as two types of architectural styles for creating web services: SOAP and REST with the focus on the latter. The main elements of SOAP architecture are described – WSDL, UDDI and SOAP protocol for message interchange between distributed systems. Because of the complex development of SOAP web services the REST architectural style emerged. In this paper, the basic architectural styles of web services are described from which the hybrid REST style emerged. Also, its data elements, connectors and components are described in detail. Complying with the six constraints imposed by REST, web services become RESTful web services. For information transfer between two web services XML and JSON formats are used, MVC design pattern is used for developing programs. In the practical part of this paper the tools and technologies required for acquiring and storing news from various websites for the purpose of enabling search over news content are described in detail. For storing and searching the news content the Elasticsearch program is used and .NET framework is used for the creation of application programming interface through which the Elasticsearch is accessed. .NET framework is also used for defining controllers necessary for handling the requests which are forwarded towards web service.



**Keywords:** .NET, Elasticsearch, NEST, REST, SOAP, web service

# ŽIVOTOPIS

Petar Marić rođen je 19. travnja 1996. godine u Osijeku. 2011. godine upisuje III. Gimnaziju u Osijeku te je završava 2015. godine. Smjer Računarstvo upisuje 2015. godine na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

# PRILOZI

CD s pismenom inačicom rada i izvornim kodom ostvarenog programskog rješenja.