Korištenje web servisa u Java aplikacijama

Sabolić, Matija

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matija Sabolić

KORIŠTENJE WEB SERVISA U JAVA APLIKACIJAMA

DIPLOMSKI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matija Sabolić

Matični broj: 45258/16–R

Studij: ***Informacijsko i programsko inženjerstvo***

KORIŠTENJE WEB SERVISA U JAVA APLIKACIJAMA

DIPLOMSKI RAD

Mentor/Mentorica:

Prof. dr. sc. Dragutin Kermek

Varaždin, rujan 2018.

Matija Sabolić

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

U ovom diplomskom radu koji se bavi Web servisima na početku će biti ukratko objašnjene najznačajnije tehnologije koje su njima prethodile. U nastavku će biti predstavljeno što tu to Web servisi te njihove osnovne značajke. Nadalje predstavljeni će biti SOAP i REST Web servisi te njihove značajke. Predstavljeni će biti i WSDL te WDAL kao dva jezika pomoću kojih se opisuju Web servisi. Obratit će se pozornost i na HTTP metode pomoću kojih se obavljaju pozivi Web servisa. Nadalje će biti objašnjeni XML i JSON kao dva najkorištenija formata zapisa Web servisa. Predstavljena će biti i sigurnost Web servisa koja u današnje doba postaje sve važnija zbog sve više napada na njih. U zadnjem dijelu ovog diplomskog rada biti će opisano programsko rješenje koje je implementirao za potrebe ovog diplomskog rada u programskom okviru Spring.

Ključne riječi: Java; Web servis; REST; SOAP; XML; JSON; Sigurnost;

Sadržaj

1. Uvod

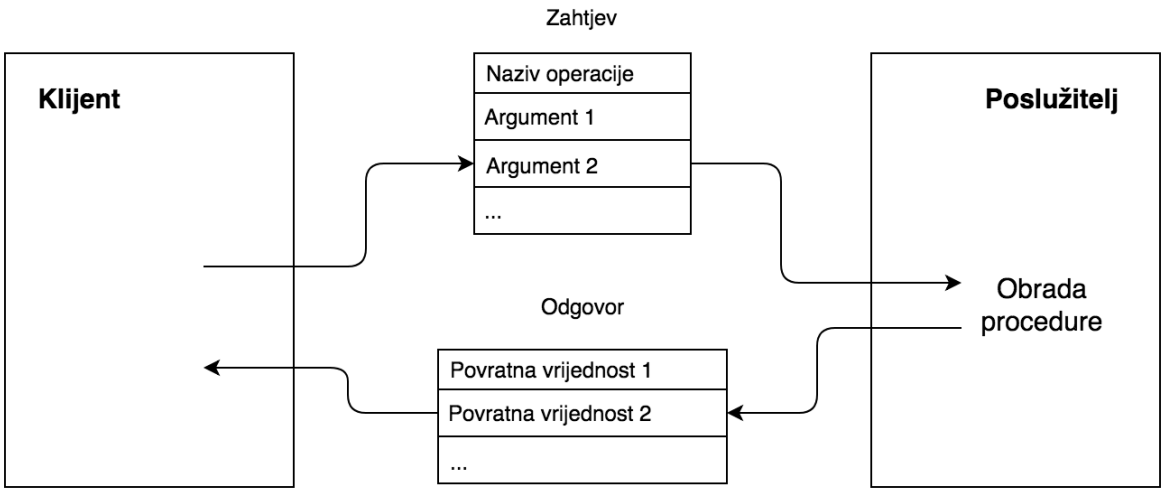
Kako u današnje vrijeme postoji mnoštvo programskih jezika i okruženja pomoću kojih je omogućen razvoj aplikacija vrlo je teško direktno komuniciranje između njih. Zbog toga su Web servisi su u današnje vrijeme postali standardizirani oblik komuniciranja između klijentske i serverske aplikacije putem WWW protokola HTTP. Web servisi nam omogućuju da više aplikacija komuniciraju i razmjenjuju podatke usprkos tome u kojem su programskom jeziku pisane. Web servis je dizajniran kao zaseban modul koji prima točno određeni skup zahtjeva te na temelju njih daje određeni skup informacija te je dostupan preko mrežno dostupne točke *(eng. endpoint)*.

Web servisi su danas neizostavni mehanizam koji se koristi prilikom implementacije većine aplikacija. Neovisno radili li se o desktop, mobilnim ili Web aplikacijama u većini slučajeva se koriste te im se iz tog razloga mora pridati velika pažnja. Iz tog razloga ovaj diplomski rad se bavi upravo Web servisima. U današnje vrijeme kada je veliki broj programskih jezika postoji mnogo načina da se Web servisi implementiraju. U praktičnom dijelu ovog diplomskog rada biti će prikazana njihova implementacija u Java programskom jeziku. Java kao programski jezik je odabran iz razloga što je jedan od danas najkorištenijih jezika u softverskom inženjerstvu.

2. Povijest povezivanja aplikacija

Tijekom 1980-ih, razvila se tehnologa koja se zove postupak udaljenog poziva *(eng. Remote procedure call, u daljnjem tekstu RPC)* te se je počela široko primjenjivati. Dvije najpopularnije vrste RPC-a su SUN RPC i DCE *(eng. Distributed computing environment)* RPC. [1]

Za aplikaciju koja ga poziva, RPC izgleda kao lokalni poziv funkcije, ali umjesto izvršavanja lokalno parametri procedure šalju se preko mreže u udaljeni program koji obrađuje proceduru. Vrijednosti koje vrati procedura, ako postoje, šalju se nazad preko mreže u aplikaciju koja je napravila poziv. RPC infrastruktura pruža osnovu distribuiranog sustava. Aplikacija koja radi zahtjev naziva se klijent, a aplikacija koja se poziva naziva se poslužitelj. [1]



Slika 1. Osnovni RPC model (prema: Bolton, 2002)

Slika 1 prikazuje osnovni RPC model. U slučaju kada klijent želi koristiti RPC parametri za pozivanje kopiraju se u međuspremnik i to u formatu koji pogodan za mrežni prijenos. Ti podaci se zatim šalju prema poslužitelju u većini slučajeva putem UDP ili TCP protokola. Zahtjev koji klijent daje je RPC poruka koja se šalje unutar paketa odabranog protokola te sadrži ime udaljene procedure koja se poziva i listu parametara. Kada poslužitelj primi klijentov zahtjev on obrađuje proceduru dobivenu u zahtjevu. Nakon što završi s obradom, poslužitelj povratne vrijednosti sprema u objekt koji je poslan nazad klijentu koji čeka njegov odgovor. Odgovor

poslužitelja je RPC poruka koja sadrži listu povratnih vrijednosti ili status pogreške *(eng. error message)* ukoliko je došlo do nje.

U nastavku će biti opisani CORBA, RMI te RMI-IIOP kao predstavnici distribuiranih sustava koji su prethodili Web servisima.

2.1. CORBA

Common Object Request Broker Architecture (u daljnjem tekstu CORBA) je standard koji je 1991. godine predstavljen od strane Object Management Group (u daljnjem tekstu OMG). CORBA je namijenjena olakšavanju komunikacije između sustava koji se implementiraju na različitim platformama. Također CORBA omogućuje komunikaciju između sustava na različitim operacijskim sustavima, različitim programskim jezicima te različitim računalnim hardverom.

2.1.1. Osnovni CORBA koncepti

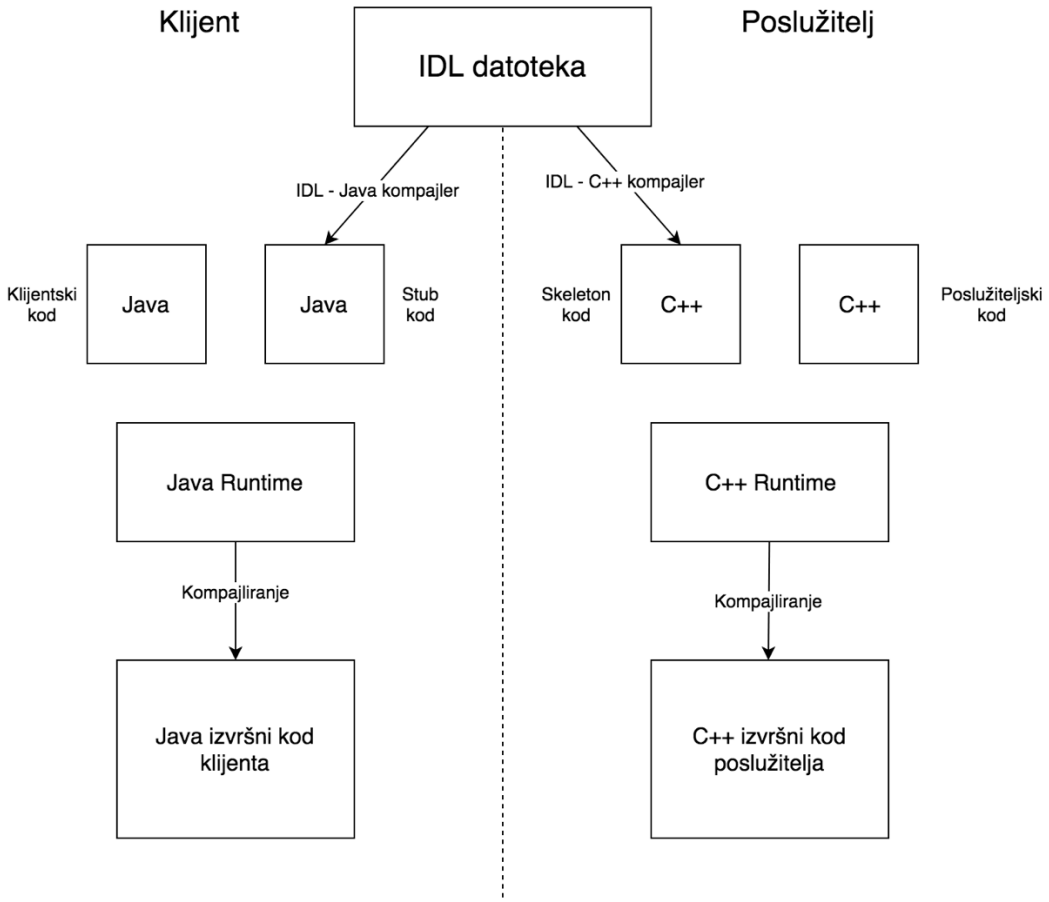
U ovom poglavlju biti će objašnjeni osnovni elementi CORBA specifikacije. Koncepti koji će biti objašnjen su:

- IDL kompilatori
- Mapiranje jezika
- Stub i skeleton
- Referenciranje objekata
- Adapteri objekata
- Posrednici objekata zahtjeva

2.1.1.1. IDL kompilatori

Definiranje jezika opisa sučelja *(eng. Interface description language, u daljnjem teksu IDL)* za sustav prvi je korak u razvoju CORBA aplikacije. Omogućuje definiranje sučelja vlastitim CORBA objektima na način koji je neovisan o platformi, neovisan o jeziku pisanja te neovisan o pojedinostima implementacije. Nakon što se definira IDL aplikacije može se odlučiti koje platforme i programski jezici će biti korišteni za klijenta i poslužitelja. [1]

IDL kompilator se koristi za mapiranje IDL-a u određeni programski jezik. Na sljedećoj slici je prikazan sustav u kojemu je klijentova implementacija u Java programskom jeziku, a implementacija poslužitelja u C++ programskom jeziku.



Slika 2. Primjer Java klijenta i C++ poslužitelja (prema: Bolton, 2002)

Slika 2 prikazuje korake koji se izvode u slučaju kada je klijentova implementacija u Java programskom jeziku a poslužiteljska implementacija u C++ programskom jeziku. Na klijentskoj strani IDL datoteka prolazi kroz IDL kompilator za Java programski jezik. IDL kompilator provodi mapiranje IDL definicija u Java programski jezik te na kraju stvara Java stub kôd kao izlaz. Stub kôd je implementacija klijenta s potrebnim kôdom kako bi se mogli koristiti pozivi Java sučelja koja su definirana u IDL datoteci. Na poslužiteljskoj strani IDL datoteka prolazi kroz IDL kompilator za C++ programski jezik. IDL kompilator provodi mapiranje IDL definicija u C++ programski jezil te na kraju stvara skeleton kôd kao izlaz. Skeleton kôd je implementacija poslužitelja s potrebnim kôdom za definiranje implementacije CORBA objekata.

2.1.1.2. Mapiranje jezika

CORBA nadopunjuje osnovnu specifikaciju s nizom dokumenata za mapiranje jezika koji određuju kako prevoditi IDL definicije u ekvivalentne konstrukcije u ciljanom jeziku. Na primjer ako je ciljani jezik C++ ili Java mapiranje jezika se izvodi na ovaj način:

- Definira kako se IDL tipovi podataka mapiraju u C++ ili Java tipove podataka
- Definira kako se IDL sučelja mapiraju u klase i kako se IDL operacije mapiraju u funkcije (C++) ili metode (Java)
- Definira kako izvršiti implementaciju CORBA objekata na strani poslužitelja [1]

2.1.1.3. Stub i skeleton

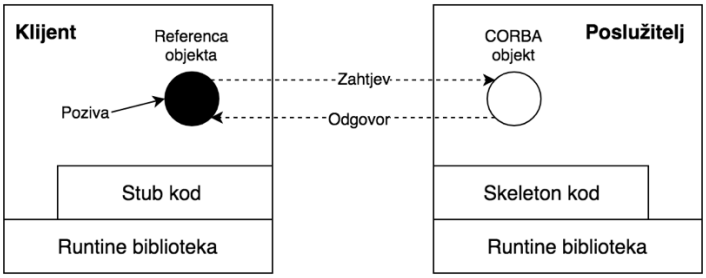
Stub i skeleton kôd generirani pomoću IDL kompilatora koriste se kako bi klijenti i poslužitelji bili svjesni definicija koje se pojavljuju u IDL datoteci. Stub kôd se koristi na strani klijenta i omogućuje klijentima pozivanje operacija na udaljenim CORBA objektima koristeći istu sintaksu kao da su lokalni objekti. Skeleton kôd se koristi na poslužiteljskoj strani kako bi omogućio poslužiteljima da implementiraju CORBA objekte. [1]

Na primjer kod Java stub kôda na slici 2 sadržane su sljedeće definicije:

- Svaki IDL tip podataka predstavlja odgovarajući Java tip podataka
- Svako IDL sučelje predstavlja odgovarajuće Java sučelje
- Svaka IDL operacija predstavlja odgovarajuću Java metodu

Skeleton kôd koji je u C++ programskom jeziku kao na slici 2 koristi se za povazivanje IDL sučelja s klasama koje su implementirane od strane programera u C++ programskom jeziku. Kod klijentovog poziva određene IDL operacije na sučelju poziva se odgovarajuća funkcija te klase.

2.1.1.4. Referenciranje objekata



Slika 3. Stvaranje udaljenog poziva putem referenciranja objekta (prema: Bolton, 2002)

Slika 3 prikazuje kako se aplikacijski kôd, stub kôd i biblioteka posrednikova zahtjeva za objekt (*eng. Object Request Broker, u daljnjem tekstu ORB)* spajaju na klijentskoj strani i kako se aplikacijski kôd, skeleton kôd i ORB biblioteka spajaju na poslužiteljskoj strani. Na klijentskoj strani CORBA sučelje za programiranje aplikacija (API) se sastoji od runtime biblioteke dajući prava ORB objektu te drugim standardnim objektima. Također CORBA API se sastoji i od stub kôda koji daje prava IDL sučeljima koja su korisnički definirana.

Klijent mora imati referencu objekta kako bi mogao izvršiti poziv CORBA objekta jer referenca objekta obuhvaća pojedinosti o lokaciji CORBA objekta. U C++ i Java programskim jezicima referenca objekta je samo objekt koji ima funkcije mapirane iz operacija odgovarajućeg IDL sučelja. Što se klijenta tiče, referencirani objekt može biti CORBA objekt. Referencirani objekt djeluje kao samostalni ili zastupljeni (proxy) objekt za CORBA objekt na strani klijenta. [1]

Međutim referencirani objekt ne implementira operaciju koja se poziva. Slika 3 prikazuje što se dešava nakon što se pozove funkcija referenciranog objekta: inicijalizira se udaljeni poziv procedure sa zahtjevom koji se šalje udaljenom poslužitelju sa parametrima operacije. Na strani poslužitelja, poziv se preusmjerava na odgovarajući CORBA objekt uz pomoć ORB biblioteke i skeleton kôda. Poslužitelj izvršava operaciju a zatim šalje odgovor koji sadrži povratnu vrijednost i parametre izlaza. Na kraju na strani klijenta, referencirani objekt vraća povratnu vrijednost i parametre izlaza aplikacijskom kôdu. [1]

2.1.1.5. Adapteri objekata

Mapiranje jezika za C++ i Javu koriste činjenicu da su C++ i Java objektno orijentirani programski jezici tako što omogućuju implementaciju klasa za CORBA objekte na isti način kao i kod običnih C++ i Java objekta.

Nakon implementiranja CORBA klase mora se naznačiti u ORB da klasa predstavlja određeno IDL sučelje. Također potreban je način kako kazati ORB-u na koji način napraviti instance od ove klase, CORBA objekte, dostupne klijentovim aplikacijama. Dio ORB-a koje je zadužen za te zadatke je adapter objekata. [1]

Bitne odgovornosti jednog adaptera objekata su:

- Omogućiti mehanizam za povezivanje C++ ili Java klase s određenim IDL sučeljem

- Upravljanje životnim ciklusom CORBA objekta. Konkretno, adapter objekta mora pružiti način aktivacije CORBA objekta (što ih čini dostupnim klijentima) i način deaktivacije CORBA objekta (što ih čini nedostupnim klijentima) [1]

Dva tipa adaptera objekata koji su najčešće korišteni su Basic Object Adapter (u daljnjem tekstu BOA) i Portable Object Adapter (u daljnjem tekstu POA). BOA dolazi iz CORBA specifikacija prije verzije 2.2. BOA ima slabu specifikaciju te su zbog toga individualni BOA dobavljači bili prisiljeni uvesti vlasnička proširenja kako bi ih bili u mogućnosti implementirati. Iz tog razloga programski kôd koji je napisan pomoću BOA nije prenosiv između ORB implementacija. POA je dodana u CORBA specifikaciji 2.2 i njezina je uloga da zamijeni BOA. POA specifikacija je detaljna i samim time omogućuje dobar stupanj prenosivosti između ORB implementacija i dodavanje mnogih poboljšanja.

2.1.1.6. ORB

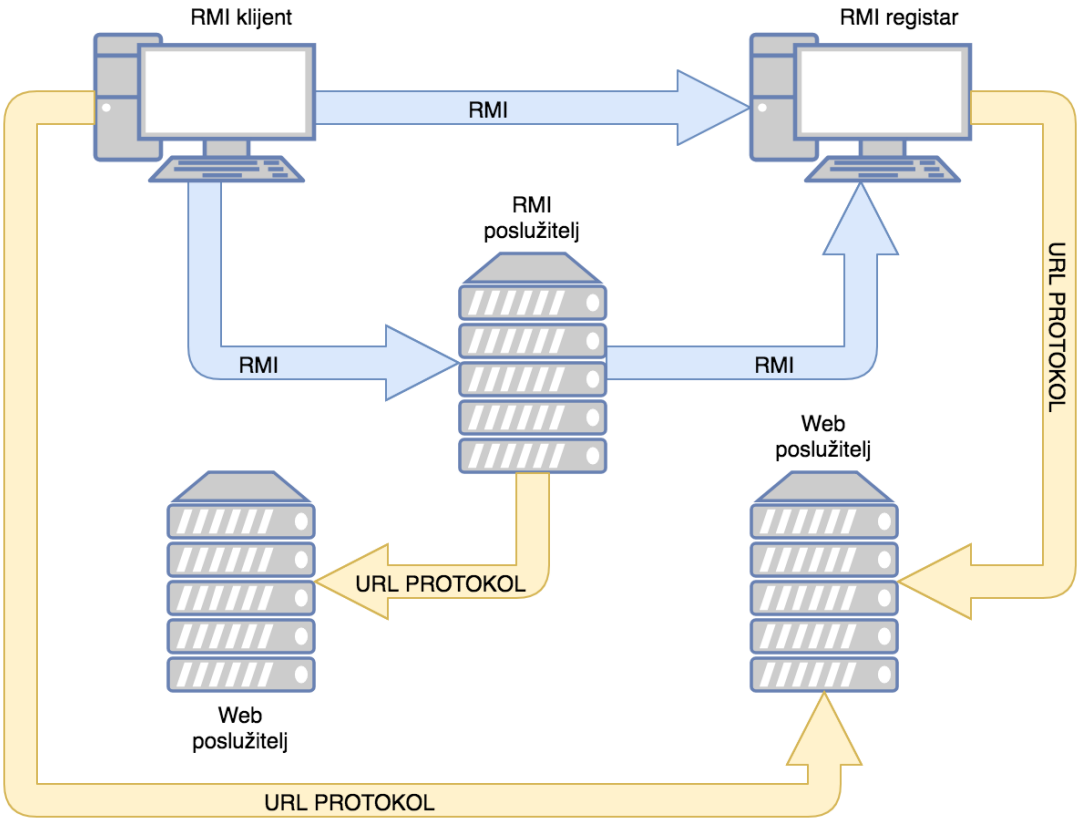
U ovoj fazi bilo bi lijepo ako bi mogli ukazati na neki entitet i reći za njega da je ORB. Međutim ORB je apstrakcija i takav entitet ne postoji. ORB je ukupna infrastruktura koja omogućuje da se izvrše udaljeni pozivi. Tipični ORB ima sljedeće glavne elemente:

- Stub i skeleton kôd koji se dobiva iz IDL-a i povezan je s aplikacijama
- Runtime biblioteku za povezivanje v aplikacijama
- Mehanizam za lociranje i aktiviranje udaljenih poslužitelja. Tipično proces demon (ili procesi) ispunjavaju ovu ulogu. [1]

2.2. RMI

Remote Method Invocation (u daljnjem tekstu RMI) je Java bazirano proširenje za RPC. Pruža mehanizam za kreiranje Java baziranih distribuiranih aplikacija. Omogućuje da je objekt iz jednog Java virtualnog stroja *(eng. Java Virtual Machine, u daljnjem tekstu JVM)* u interakciji s objektom iz druge JVM pozivajući metode tog objekta. Zbog toga se aplikacija izgrađena pomoću RMI smatra aplikacijom koja se može izvoditi na više JVM uređaja. [2]

RMI omogućuje komunikaciju između aplikacija koje se izvršavaju na različitim poslužiteljima i povezuju se preko mreže pomoću objekata koji se zovu stub i skeleton. Ova komunikacijska arhitektura čini distribuiranu aplikaciju kao skupinu objekata koji komuniciraju preko udaljenog povezivanja.



Slika 4. Komunikacija RMI klijenta i RMI poslužitelja preko RMI registra (prema: PATTAMSETTI, 2017)

Slika 4 prikazuje postupak referenciranja udaljenih objekata. Poslužitelj poziva RMI registar da registrira odnosno da pridruži naziv udaljenom objektu. Klijent traži udaljeni objekt preko njegovog naziva u RMI registru poslužitelja i tada on može pozvati njegove metode. Također vidi se da RMI sustav koristi postojeći Web poslužitelj za učitavanje stub-ova od poslužitelja do klijenta i klijentske stub-ove do poslužitelja kada je to potrebno.

U nastavku će biti nabrojano što se sve treba uzeti u obzir kada se dizajnira aplikacija pomoću RMI:

- Besprijekoran način povezivanja s objektima koji su kreirani na više JVM uređaja
- Osiguravanje da se poziv udaljene metode može jednostavno integrirati s općom logikom programiranja bez dodatnih IDL-ova zadržavajući većinu Java semantike
- Postavljanje sposobnosti razlikovanja distribuiranog i lokalnog objektnog modela
- Pomaganje u izgradnji pouzdanih aplikacija uz održavanje Java sigurnosti.

- Pružanje podrške za korištenje više transportnih protokola, različite referentne semantike kao što su upornost i lijena aktivacija, te različiti mehanizmi povezivanja
- Rješavanje problema kao što su rad u drugačijim memorijskim prostorima, prolazak parametara, spajanje podataka i drugi kvarovi RPC-a
- Rješavanje eventualno dodatnih problema [2]

2.2.1. RMI terminologija

U nastavku će biti objašnjeni neki od glavnih termina kada je u pitanju RMI:

- **Udaljeni objekt** – Ovo je objekt u specifičnom JMV čije metode su izložene kako bi se mogle pozivati pomoću drugog programa koji je smješten na drugi JVM
- **Udaljeno sučelje** – Ovo je Java sučelje koje definira metode koje postoje u udaljenom objektu. Udaljeni objekt može implementirati više od jednog udaljenih sučelja za prihvaćanje više ponašanja udaljenih sučelja.
- **Stub** – Ovo je Java objekt koji služi kao ulazna točka za klijentski objekt. Postoji u klijentskom JVM i bavi se rukovanjem s udaljenim objektom. Ako bilo koji objekt pozove metodu od stub objekta oni odradi sljedeće korake:
    - Inicijalizira vezu s udaljenim JVM
    - Priprema *(eng. marshal, piše i prenosi)* parametre do udaljenog JVM
    - Čeka odgovor od udaljenog objekta te priprema *(eng. unmarshal, čita)* vraćenu vrijednost ili pogrešku te zatim odgovara pozivatelju s tom vrijednošću ili pogreškom
- **Skeleton** – Ovo je objekt koji se ponaša kao prolaz *(eng. gateway)* na poslužiteljskoj strani. Djeluje kao udaljeni objekt pomoću kojega korisnički objekti komuniciraju sa stubom. To znači da se svi zahtjevi koji dolaze iz udaljenog klijenta usmjeravaju pomoću njega. Ako skeleton primi zahtjev on odradi ove korake:
    - Čita parametar koji je poslan u udaljenu metodu
    - Poziva metodu udaljenog objekta
    - Vraća rezultat nazad pozivatelju [2]

2.2.2. Distribuirano računalstvo pomoću RMI

Kada se govori o pozivanju metoda i vraćanju rezultata iz istih udaljeni objekti su slični lokalnim objektima. Udaljeni objekt se može pretvoriti u bilo kojem udaljenom sučelju koje podržava njegovu implementaciju dok se operator `instanceof` može upotrijebiti za provjeru tipa udaljenog ili lokalnog objekta.

Iako postoje sličnosti između udaljenih i lokalnih objekata koje su prije navedene, njihovo ponašanje je drugačije. Klijenti udaljenih objekata mogu komunicirati samo s udaljenim sučeljima, a ne s njihovim klasama implementacije. Klijenti lokalnih objekata mogu komunicirati s sučeljima i klasama implementacije. RMI se prosljeđuje samo vrijednošću, dok se poziv lokalne metode može obaviti prosljeđivanjem reference. Obavezne udaljene iznimke *(eng. remote exceptions)* koje se mogu pojaviti moraju biti definirane kod lokalnog sustava.

U nastavku su navedeni bitni koraci koji se moraju pratiti ukoliko se kreće u razvoj distribuirane aplikacije uz pomoć RMI:

1. Dizajnirati i implementirati komponentu koje neće samo biti uključena u distribuiranu aplikaciju, također dizajnirati i implementirati lokalnu komponentu
2. Osigurati da komponente koje sudjeluju u RMI pozivima budu dostupne u svim mrežama
3. Uspostaviti mrežnu vezu između aplikacija koje trebaju stupiti u interakciju preko RMI [2]

U prethodnom popisu najvažniji korak koji se treba pažljivo provesti definira pravu arhitekturu u aplikaciji s jasnom razlikom između komponenti koje djeluju kao Java objekti dostupni na lokalnim JVM i onih koji su dostupni udaljeno. U nastavku su navedeni detaljni koraci provedbe:

1. **Definicija udaljenog sučelja** – Svrha definiranja udaljenog sučelja je deklarirati metode koje bi trebale biti dostupne za pozivanje od strane udaljenog klijenta. Programiranje sučelja umjesto implementacije komponenata bitan je princip dizajna koji su prihvatili svi moderni Java programski okviri pa tako i Spring. U istom uzorku definicija udaljenog sučelja ima važnost i kod RMI dizajna.
2. **Implementacija udaljenih objekata** – Java dopušta da jedna klasa implementira više sučelja odjednom. To pomaže udaljenim objektima implementirati jedno ili više udaljenih sučelja. Klasa udaljenog objekta će možda morati implementirati druga lokalna sučelja i metode za koje je odgovorna. Izbjegavajte dodavanje složenosti u ovom scenariju u smislu na koji način argumenti i vrijednosti koje se vraćaju u takvim metodama komponente moraju biti napisani.
3. **Implementacija udaljenih klijenata** – Klijentski objekti koji komuniciraju s udaljenim poslužiteljskim objektima mogu se napisati nakon što su udaljena sučelja pažljivo definirana čak i nakon što su udaljeni objekti razvijeni. [2]

2.3. RMI-IIOP

Java RMI-IIOP kombinira najbolje značajke Java RMI tehnologije s najboljim značajkama CORBA tehnologije. Poput Java RMI, RMI-IIOP ubrzava implementaciju distribuiranih aplikacija dopuštajući programerima da rade potpuno u Java programskom jeziku. Kada se koristi RMI-IIOP kako bi se izradile Java bazirane distribuirane aplikacije nema posebnog IDL-a ili mapiranja koje se treba naučiti. Baš kao i Java RMI, RMI-IIOP pruža fleksibilnost dopuštajući programerima da prenose bilo koji serijalizirani Java objekt između komponenata aplikacije. Poput CORBE, RMI-IIOP koristi IIOP ako svoj komunikacijski protokol. IIOP olakšava nasljeđivanje aplikacije i integraciju platforme dopuštajući aplikacijskim komponentama napisanim u C++, Smalltalk i drugim CORBA podržanim programskim jezicima da komuniciraju s komponentama koje se izvode na Java platformi. [3]

Pomoću RMI-IIOP programeri mogu pisati udaljena sučelja u Java programskom jeziku i implementirati ih samo koristeći Java tehnologije i Java RMI API. Ta se sučelja mogu implementirati u bilo kojem drugom programskom jeziku koji je podržan od strane OMG mapiranja i za koje je dobavljač isporučio ORB. Slično tome klijenti mogu biti napisani u drugim programskim jezicima koristeći IDL koji je izveden iz udaljenog sučelja koje je bazirano na Java programskom jeziku. Koristeći RMI-IIOP objekti se mogu biti zadani preko reference ili preko vrijednosti koristeći IIOP. [3]

RMI-IIOP može se preuzeti u sklopu Java 2 platforme, standardne edicije u verzijama 1.3 i 1.4. RMI-IIOP IDL kompilator smješten je u `bin` direktoriju J2SDK instalacije. RMI-IIOP IDL kompilator u mogućnosti je generirati stub i skeleton kôd te veze za udaljene objekte koristeći IIOP protokol. RMI-IIOP IDL kompilator generira prije navedene dijelove pomoću opcije `-iiop`. Pomoću navedenog kompilator također je moguće generirati OMG IDL za CORBA programiranje na Java platformi.

3. Web servisi

Web servis je u većini slučajeva proizvod u obliku funkcije koja se može ponovno koristiti *(eng. reusable function)* izgrađen od strane jedne osobe ili kompanije i postavljen unutar mreže kako bi ga druge osobe ili kompanije mogle koristiti. [4]

Koncept Web servisa sve više raste i razvija se novi uzbudljivi model poslovanja. Sada je slučaj da koncept Web servisa eksplodira na nova područja i proširuje svoje mogućnosti. Osnovni cilj Web servisa koji je na visokoj razini je da se unaprijedi distribuirano računalstvo (gdje je logika aplikacije razdvojena u male logičke dijelove i radi na više računala ili uređaja). Praktični razlog za izgradnju Web servisa je omogućiti kompanijama da izgrade male, ponovno iskoristive i samo opisujuće metode koje mogu i drugi koristiti. Web servis sam po sebi je dio kôda koji može biti pozvan od strane druge aplikacije ili udaljenog procesa. Međutim ako bi programer u bliskoj budućnosti mogao spojiti zajedno dovoljan broj Web servisa on bi mogao implementirati većinu značajki potrebnih za podršku cijeloj Web aplikaciji. Praktični cilj Web servisa je da se kompanijama omogući da svoje poslovanje koncentriraju samo na potrebe svoje osnovne djelatnosti te da pozivaju Web servise kako bi nadopunili svoje osnovne djelatnosti. [4]

Razlika Web stranice u odnosu na Web servis je u korištenju XML-a ili JSON-a pomoću kojih se definiraju i kontroliraju podaci koji se šalju na Web servis ili se dobivaju od strane Web servisa. Web servis je definiran pomoću zbirke alata i specifikacija kojima se programeri služe prilikom njihove izrade. Kako svi koriste istu skup standardnih specifikacija Web servisi rade kako su i zamišljeni.

Kada se govori o Web servisima potrebno je definirati pojmove klijenta i poslužitelja u njihovom kontekstu. Poslužitelj Web servisa je svaki uređaj koji odgovara na dobivene zahtjeve. Poslužitelj upravlja resursima na način da kontrolira izlaz podataka putem odgovora Web servisa. Klijent Web servisa je svaka aplikacija, korisnik ili Web preglednik koji poziva poslužitelja dajući mu zahtjeve za obavljanjem određenih operacija. Klijent Web servisa se oslanja na poslužitelja Web servisa za dobivanje informacija putem odgovora.

3.1. Osobine i vrste

Glavne značajke što se tiče kreiranja i korištenja Web servisa su:

- Dostupni su preko standardnih internetskih protokola kao što su HTTP ili SMTP
- Distribuirani su, što znači da je Web servis u većini slučajeva smješten na drugom poslužitelju od aplikacije koja ga koristi
- Mogu biti centralizirani kao jedan izvor. Što znači da možete implementirati nešto jedanput i pristupiti tome mnogo puta iz drugih projekata. Drugim riječima Web servisi omogućuju povećanje ponovne iskoristivosti kôda.
- Jedan Web servis nije potpuna aplikacija već samostalna funkcija koju mogu pozvati mnoge različite aplikacije
- Web servis može biti samo opisujući. To omogućava kompanijama i aplikacijama da pronađu i koriste Web servise pomoću automatiziranih procesa i internetskih registara [4]

Sve ove značajke se zbrajaju kako bi se kreirao ponovno upotrebljiva komponenta Web servisa koja može pružati svoju funkcionalnost preko interneta. [4]

Prednosti kreiranja i korištenja Web servisa dolaze iz distribuirane prirode cjelokupnog sustava. Prednosti su sljedeće:

- Logika može biti razdijeljena na male ponovno upotrebljive dijelove kôda
- Programski kôd se može koristiti u mnogim različitim aplikacijama. Npr. Web servisu napisanom u ASP.NET-u može se pristupiti preko JSP stranice.
- Programski kôd se može registrirati tako da mnoge različite organizacije mogu koristiti jedan Web servis. Rezultat toga kreator Web servisa ne mora komunicirati sa svakim kupcem
- Razvijaju se standardni protokoli, API-i (standardni kôd) i alati (razvojni paketi za programiranje) kako bi omogućili programerima da izgrađuju i pristupaju Web servisima. [4]

Nedostaci Web servisa uključuju sljedeće:

- Mora se izgraditi drugi softverski sloj za korištenje Web servisa. Taj novi sloj znači da se mora pažljivo razmotriti arhitektura aplikacije i usluge u sustavima koji koriste Web servis

- Pristup Web servisima putem interneta uzrokuje probleme sa sigurnošću i brzinom dizajneru aplikacije. Međutim može se povećati sigurnost Web servisa na štetu njegove brzine [4]

Kao što je čest slučaj u računalnoj industriji, potrebno je nekoliko iteracija kako bi se te stvari ispravile. Kada su u pitanju Web servisi to je osobito istinito jer su Web servisi nastali na temelju starijih ideja. Njihov dizajn pomaže u smanjenju vremena razvoja i potiče ponovnu iskoristivost programskog kôda. [4]

Postoje dva tipa Web servisa, Simple Object Access Protokol (u daljnjem tekstu SOAP) i Representational State Transfer (u daljnjem tekstu REST) Web servis. U nastavku će biti opisane pojedinosti od svakoga te će biti dana usporedba ta dva servisa.

SOAP je protokol baziran na XML formatu zapisa. Jedna od najvećih njegovih prednosti je sigurnost koju pruža. SOAP pruža omotnicu *(eng. envelope)* uz pomoć koje se šalju poruke preko mreže putem HTTP protokola. Poruke koje se šalju unutar SOAP omotnice su u XML formatu zapisa. Najjednostavnije rečeno SOAP je tehnika za slanje XML zahtjeva te primanje rezultata u XML formatu putem HTTP mrežnog protokola.

Primjer SOAP Web servisa se može bazirati na studentima nekog fakulteta. Ako klijent Web servisa želi dohvatiti podatke određenog studenta slanjem samo matičnog broja studenta, on to može napraviti preko Web servisa. Jedino što mu je potrebno je da zna koji URI pozvati i na koji način njegov zahtjev mora biti strukturiran. Te podatke klijent može pronaći u WSDL datoteci. Svaki SOAP Web servis ima WSDL *(eng. Web Service Description Language)* koji je u XML formatu i označava jezik za opisivanje Web servisa. WSDL pruža opis svih dostupnih metoda u Web servisu zajedno za strukturama zahtjeva i odgovora.

Kako softverska industrija napreduje tako se može vidjeti da se SOAP Web servisi danas većinom koriste u poslovnim aplikacijama, uglavnom u naslijeđenom programskom kôdu. Danas je većina Web servisa bazirana na REST tehnologiji. REST Web servisi se bave resursima za razliku od akcija na koje se bazira SOAP. REST Web servisi smještaju resurse pomoću URI-a i ovisno o vrsti HTTP metode koja se koristi izvršava se određena radnja nad zadanim resursom.

Na primjeru studenata nekog fakulteta, u REST Web servisu URI oblika http://example.com/students/student/07 može biti upotrijebljen:

- Da se dobe podaci studentu pozivanjem GET metode. REST Web servis će vratiti podatke o studentu s matičnim brojem 07
- Da se ažuriraju podaci o studentu pozivanjem PUT metode zajedno s novim vrijednostima za studenta s matičnim brojem 07
- Da se obrišu podaci o studentu s matičnim brojem 07 pozivanjem DELETE metode

Neke osnovne razlike između SOAP i REST Web servisa biti će navedene u sljedećoj tablici:

Tablica 1. Osnovne razlike između SOAP i REST servisa

REST	**SOAP**
REST je stil softverske arhitekture	SOAP je protokol ili set standarda
REST može koristiti SOAP jer je koncept i može koristiti protokol kao što je HTTP, SOAP, itd.	SOAP ne može koristiti REST jer je sam po sebi protokol
REST upotrebljava URI kako bi predstavio poslovnu logiku. Kako REST radi na temelju vrste HTTP zahtjeva, tako isti URI može raditi više od jedne vrste operacija	SOAP koristi servisno sučelje kao bi predstavio poslovnu logiku
REST ne definira previše standarda	SOAP definira standarde koji se moraju strogo slijediti
REST nasljeđuje sigurnosne mjere iz temeljnim transportnih protokola	SOAP definira svoj vlastiti sigurnosni sloj
REST prihvaća različite formate podataka kao što su običan tekst, HTML, XML, JSON	SOAP radi samo sa XML

(prema: https://www.studytonight.com/rest-web-service/types-of-webservices , Pristupljeno: 01-kol-2018)

4. SOAP Web servisi

SOAP je započeo kao Simple Object Access Protocol, kojega su razvili Microsoft, Developmentor i Userland. Nakon toga IBM i Lotus pridonijeli preglednoj specifikaciji koja je rezultirala SOAP verzijom 1.1 objavljenom u travnju 2000. godine. Ova specifikacija je bila široko prihvaćena diljem industrije i osnovala je nekoliko otvorenih izvora interoperabilnih implementacija. [5]

Autori su u svibnju 2000. godine SOAP 1.1 specifikaciju izložili W3C-u *(eng. World Wide Web Consortimu)* gdje je tretiran kao ulazni dokument za osnivanje XML Protocol Working Group u rujnu 2000. godine. Ova radna skupina trebala je donijeti standardizaciju SOAP-a. [5]

Ljudi su bili zbunjeni u vezi značenja kratice SOAP jer se koristilo nekoliko popularnih interpretacija. Radna skupina W3C konačno je u lipnju 2003. godine donijela preporuku naziva SOAP 1.2 Recommendation. Preporuka se sastojala od osnovnog SOAP 1.2 te okvira za razmjenu poruka i datoteka. Radna skupina je također razvila specifikacije za MTOM *(eng. Message Transmission Optimization Mechanism)* i XOP *(eng. XML-binary Optimized Packaging)*. [5]

SOAP specifikacija uvodi skup pojmova koji opisuju protokol zajedno s prijenosom i primanjem inkapsuliranih podataka. Na početku je važno da definiramo neke pojmove koji se u nastavku koriste za detaljnije opisivanje rada SOAP-a.

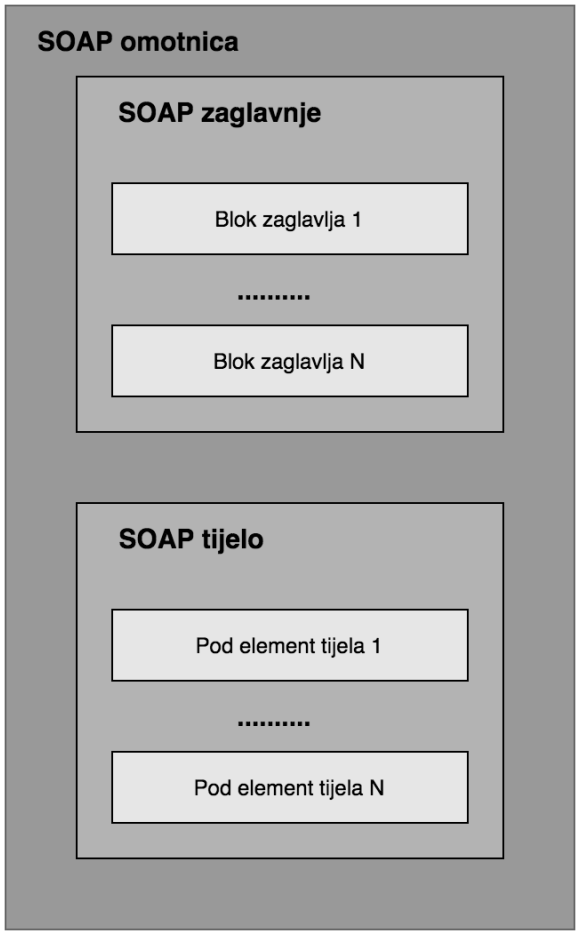
4.1. Osnovi SOAP pojmovi

SOAP se sastoji od skupa konvencija koje specificiraju format SOAP poruka. Također te konvencije specificiraju i skup pravila koja su zadužena za regulaciju obrade podataka dok poruka prolazi kroz put SOAP poruke. Konvencije opisuju na koji način se gradi SOAP poruka te koje interakcije imaju SOAP čvorovi obrađujući SOAP poruku duž puta SOAP poruke.

SOAP poruka je je osnovna jedinica komunikacije između SOAP čvorova. Sastoji se od SOAP omotnice koja sadrži nula ili više SOAP zaglavlja. SOAP zaglavlja usmjerena su na bilo koji SOAP prijamnik koji bi mogao biti na putu SOAP poruke. SOAP omotnica također sadrži SOAP tijelo koje sadrži podatke o korisničkom sadržaju ili poslovne informacije. SOAP tijelo

može sadržavati npr. zahtjev i ulazne podatke koje servis treba obraditi. Prilikom obrade SOAP poruke, SOAP čvor može generirati iznimku. Ako se taj slučaj dogodi SOAP čvor vraća SOAP poruku koja sadržava SOAP iznimku. [5]

Prethodno opisani elementi prikazani su na sljedećoj slici.

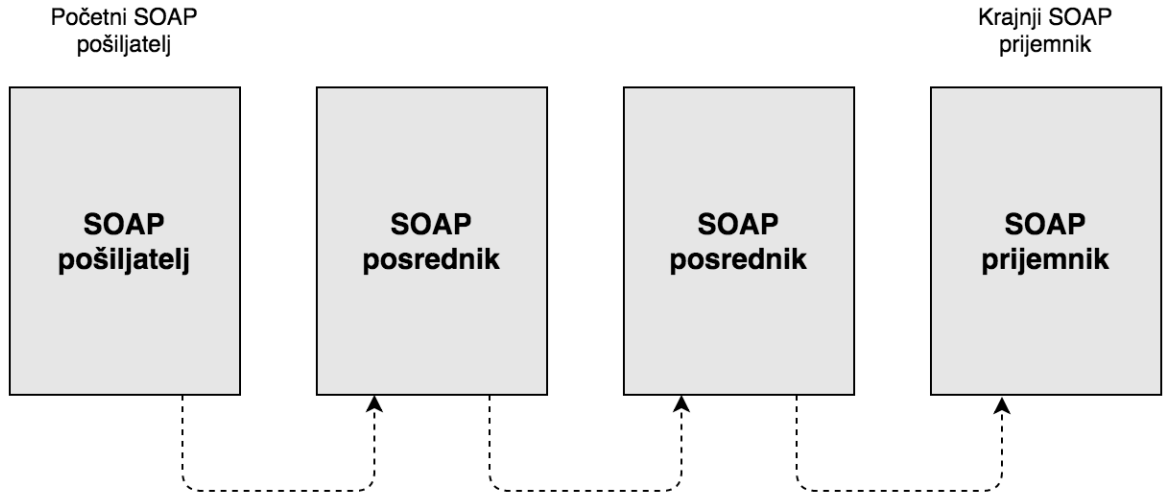


Slika 5. Ugniježđeni elementi SOAP poruke (prema: Weerawarana, 2005)

SOAP čvor je implementacija pravila za obradu koja su opisana u SOAP specifikaciji. SOAP čvor može prenijeti, primiti, odrađivati i prosljeđivati SOAP poruku. Iako SOAP čvor implementira model obrade SOAP-a, može pristupiti i svim servisima koje bi mogao pružati mrežni protokol. To čini pomoću SOAP spajanja *(eng. SOAP binding)* koji specificira pravila za prijenos SOAP poruke na vrhu nekog drugog temeljnog mrežnog protokola. Ti protokoli prijenosa mogu biti i drugi standardni kao što su HTTP, SMTP ili TCP. Međutim oni mogu uključivati i vlasničke protokole kao što je IBM WebSphereMQ.m. [5]

SOAP čvorovi zaduženi su za primanje i slanje SOAP poruka. SOAP čvor koji je zadužen za prijenos poruke naziva se SOAP pošiljatelj. SOAP čvor koji je zadužen za primanje poruke naziva se SOAP prijamnik. SOAP čvorovi koji su zaduženi za primanje i prijenos poruka nazivaju se SOAP posrednici. SOAP pošiljatelj koji gradi SOAP poruku naziva se početni SOAP pošiljatelj. Odredišni SOAP prijamnik naziva se krajnji SOAP prijamnik. Krajnji SOAP prijamnik zadužen je za obradu poruke koja se nalazi u SOAP tijelu.

Početni SOAP pošiljatelj na početku gradi SOAP poruku koja zatim prolazi kroz različite SOAP posrednike prije nego što dođe na krajnji SOAP prijamnik. Taj skup čvorova kroz koje prolazi SOAP poruka naziva se put SOAP poruke. U nekim slučajevima može se desiti da SOAP poruka neće doći do krajnjeg SOAP prijemnika jer određeni SOAP posrednik pogrešno prati obradu iznimaka. Put SOAP poruke prikazan je na sljedećoj slici.



Slika 6. Put SOAP poruke (prema: Weerawarana, 2005)

4.2. SOAP model obrade

Specifikacija SOAP poruke izražena je kao XML Infoset. To znači da SOAP pošiljatelj mora stvoriti XML Inforset kojega SOAP prijemnik može rekonstruirati. Kako bi se ono moglo napraviti SOAP pošiljatelj mora serijalizirati XML Infoset na način da SOAP prijemnik može koristiti za rekonstrukciju originala. Standardni način serijalizacije je korištenje XML 1.0 sintakse, ali specifikacija omogućuje druge, potencijalno više optimizirane prikaze za učinkovitiji mrežni promet. [5]

Svi primjeri koji će biti navedeni u nastavku koriste standardnu XML 1.0 sintaksu. Sljedeći primjer prikazuje osnovnu SOAP poruku i njezine ugniježđene elemente:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
        <oh:zaglavljeNarudzbe xmlns:pns="http://example.com/narudzba/zaglevlje">
            <oh:narucioc>FOI</oh:narucioc>
            <oh:prioritet>Visok</oh:prioritet>
            <oh:datumVrijeme>2018-08-15T09:45:00+00:00 </oh:datumVrijeme>
        </oh:zaglavljeNarudzbe >
    </env:Header>
    <env:Body>
        <oi:stavkaNarudzbe xmlns:env="http://example.com/narudzba/stavka">
            <oi:naziv>USB</oi:naziv>
            <oi:kolicina>100</oi:kolicina>
            <oi:cijena>3000</oi:cijena>
        </oi:stavkaNarudzbe>
    </env:Body>
</env:Envelope>
```

Element koji se nalazi kao omotač svega je element `env:Envelope` koji uključuje prostor imena za SOAP. U njemu su obuhvaćena dva elementa koja definira SOAP. To su elementi env:Header i env:Body. SOAP specifikacija ne definira njihov sadržaj. Elementi su specifični za aplikaciju koja stvara i obrađuje SOAP poruku. Međutim SOAP specifikacija definira kako SOAP čvor obrađuje te elemente. [5]

Element env:Header je opcionalan što se tiče SOAP poruke, Ali je uključen u prethodni primjer kako bi se prikazala njegova upotreba. SOAP zaglavlje je mehanizam za proširenje koji omogućuje prijenos informacija unutar SOAP poruke koje nisu dio poslovnih informacija. U primjeru koji je prethodno prikazan env:Header sadrži element dijete koji sadrži skup informacija o narudžbi koje nisu stavke narudžbe. Element dijete prethodno opisan se naziva blok zaglavlja sa SOAP specifikacijom. Također zaglavlje se može sastojati od više blokova zaglavlja. Blok zaglavlja u prethodnom primjeru ima vlastiti XML prostor imena u kojemu se pojavljuju elementi oh:narucioc, oh:prioritet i oh:datumVrijeme. Element oh:narucioc označava naručioca narudžbe, element oh:prioritet označava prioritet narudžbe, te element oh:datumVrijeme označava vrijeme slanja narudžbe.

SOAP specifikacija omogućava da blokovi zaglavlja budu usmjereni na određene SOAP čvorove za obradu dok poruka putuje kroz SOAP put. Kreiranjem novih zaglavlja mogu se stvoriti interoperabilni protokoli na temelju SOAP-a. [5]

Element env:Body je obavezan u SOAP poruci. Sadrži informacije koje se prenose od početnog SOAP pošiljatelja do krajnjeg SOAP prijamnika. Informacije se mogu prenijeti unutar env:Header i env:Body, te odluka koje informacije se smještaju u koji element ostaje na poslovnoj aplikaciji ili na arhitektu sustava. U većini poslovnih okružena env:Body sadrži informacije koje su nužne za određenu aplikaciju, tj. informacije koje su obrađene od strane krajnjeg SOAP prijamnika. Ostale informacije koje su potrebne se prenose u env:Header.

4.3. SOAP iznimke

U slučaju kada dođe do iznimke prilikom obrade SOAP poruke, SOAP specifikacija pruža model za rukovanje istima. Upravljanje SOAP iznimkama razlikuje razloge nastanka iznimke i mehanizme putem kojih se signalizira nastanak iznimke. Signalizacija ovisi o osnovnom transportnom mehanizmu te je opisana u SOAP specifikaciji.

Informacije o SOAP iznimki nalaze se unutar env:Body elementa. Sve iznimke, aplikacijske ili neke druge moraju koristiti strukturu koja je prikazana u sljedećem primjeru:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:flt="http://example.com/faults">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Receiver</env:Value>
        <env:Subcode>
          <env:Value>flt:BadValue</env:Value>
        </env:Subcode>
      </env:Code>
      <env:Reason>
        <env:Text>A Fault occurred</env:Text>
      </env:Reason>
      <env:Detail>
        <flt:MyDetails>
          <flt:Message>Something went wrong at the Receiver</flt:Message>
          <flt:ErrorCode>1234</flt:ErrorCode>
        </flt:MyDetails>
      </env:Detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

SOAP iznimke prikazuju se unutar jednog env:Fault elementa koji je dijete od env:Body elementa. Element env:Fault mora sadržavati dva elementa koja su mu djeca a to su env:Code i env:Reason. Također može još sadržavati i elemente env:Detail, env:Node i env:Role.

Element env:Code mora sadržavati dijete element env:Value koji sadrži jedan od pet kodova SOAP iznimki koji su definirani u SOAP specifikaciji. Kodovi koji su specificirani su sljedeći:

- **VersionMismatch** – Poruka se ne podudara sa SOAP verzijom
- **MustUnderstand** – Ciljani čvor ne razumije zaglavlje u poruci koja sadržava atribut „mustUnderstand“
- **DataEncodingUnknown** – Ciljani čvor ne razumije enkodiranje sadržaja poruke
- **Sender** – Poruka je bila pogrešno formatirana kada ju je primio čvor za obradu
- **Receiver** – Čvor primatelja ili krajnji SOAP prijemnik nije mogao obraditi poruku [5]

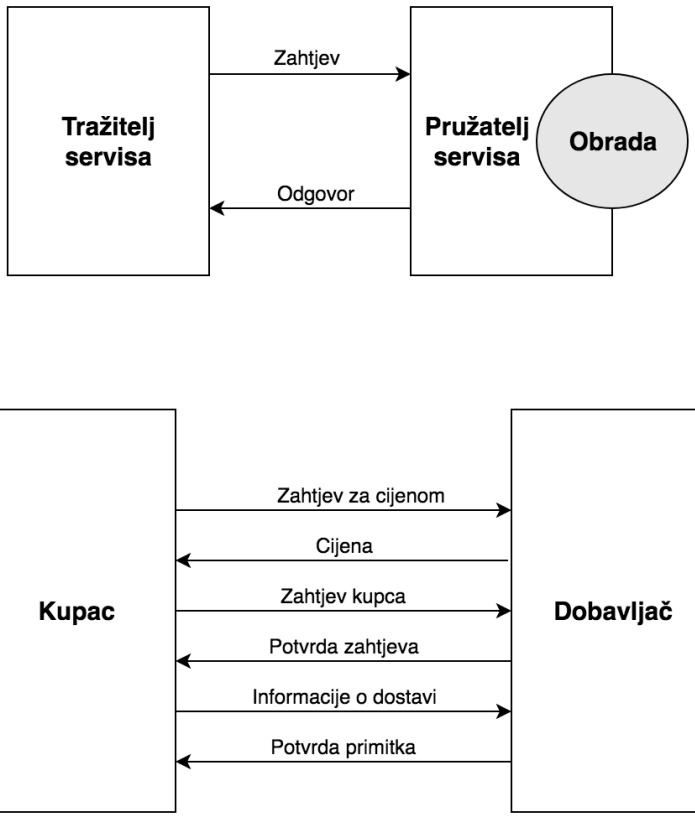
Element env:Code se dodatno može proširiti sa elementom env:Subcode koji je opcionalan. [5]

Element env:Reason unutar strukture iznimke namijenjen je čovjeku čitljivom opisu a ne elementu koji se automatski obrađuje. Ako je potrebno, element env:Text može dodatno sadržavati jezične atribute kako bi se omogućilo korištenje alternativnih nacionalnih jezika za opis dobivene iznimke. [5]

Element env:Detail koji je opcionalan može sadržavati elemente koji su specifični za neki prostor imena kako bi se pružile dodatne informacije o generiranoj iznimci. Također elementi env:Node i env:Role mogu sadržavati URI čvora koji je generirao iznimku zajedno sa njegovom ulogom. [5]

4.4. Uzorci razmjene poruka

SOAP specifikacija opisuje jednosmjeran način slanja poruka za prijenos informacija između početnog SOAP pošiljatelja i krajnjeg SOAP prijemnika. Informacije su opisane u obliku SOAP Infoseta kojega je stvorio početni SOAP pošiljatelj i ponovno stvorio krajnji SOAP prijemnik. Za praktičnu primjenu ovaj jednosmjerni model može se proširiti kako bi osigurao korisnije informacije između Web servisa. Te interakcije se zovu uzorci razmjene poruka. Na sljedećoj slici prikazana su dva tipa uzorka koja se koriste, uzorak zahtjev – odgovor i uzorak duge razmjene poruka.



Slika 7. Uzorci razmjene poruka (prema: Weerawarana, 2005)

4.4.1. Uzorak zahtjev – odgovor

Većina Web servisa djeluje na principu uzorka zahtjev – odgovor. U tom uzorku, servis šalje zahtjev davatelju servisa. Davatelj servisa zatim obrađuje poruku i vraća odgovor na servis zahtjev. Zbog bezbrižnosti Web servisa, ne postoji stanje „sesije“ kao u drugim mrežnim protokolima kao što je HTTP. Zbog toga servis zahtjev treba neki način na koji će odgovor povezati na zahtjev koji ga je pozvao. Postoje dva moguća rješenja za ovaj problem:

- Korelacija na temelju aplikacije
- Korelacija na temelju posredničkog sloja [5]

U prvom rješenju korelacija zahtjeva i odgovora provodi se na razini aplikacije. Ovo se oslanja na dizajn poslovnih poruka koje će se prenositi u SOAP env:Body elementu. [5]

Na primjer aplikacija za obradu narudžbenica može biti dizajnirana na način da uključuje jedinstveni broj narudžbenice kao element u shemi dokumenta. Početna aplikacija SOAP pošiljatelja postavlja vrijednost tog elementa i šalje ga u drugi servis koji djeluje kao krajnji

SOAP prijemnik. Kada servis narudžbenice kreira odgovor stavi broj narudžbenice iz zahtjeva u dokument. Servis narudžbenice tada postaje početni SOAP pošiljatelj koji šalje odgovor na izvorni servis koji u ovom slučaju djeluje kao krajnji SOAP prijemnik koji prima odgovor. Izvorni servis na ovaj način može povezati većinu zahtjeva i odgovora.

U drugom rješenju element env:Header u SOAP poruci sadrži informacije o korelaciji. U ovom slučaju zaglavlje bi moglo biti posebno dizajnirano za određeni skup aplikacija ili bi elementi zaglavlja koji sadrže informacije o poruci morali biti definirani kao dio adresiranja Web servisa. [5]

Ukoliko je korišten način adresiranja Web servisa, poruka zahtjeva bi mogla imati sljedeće informacije u zaglavlju:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xml.org/ws/2004/03/addressing">
    <env:Header>
        <wsa:MessageID>uuid:aaaabbbb-cccc-dddd-eeee-ffffffffffff</wsa:MessageID>
        <wsa:ReplyTo>
            <wsa:Address>http://example.com/requestAddress</wsa:Address>
        </wsa:ReplyTo>
    </env:Header>
    <env:Body>
    ...
  </env:Body>
</env:Envelope>
```

Poruka zahtjeva ima jedinstveni identifikator koji pruža element ws:MessageID. Također ima i element ws:ReplyTo koji označava gdje treba poslati odgovor na zahtjev. Poruka odgovora koju generira davatelj servisa strukturirana je na sljedeći način:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xml.org/ws/2004/03/addressing">
    <env:Header>
        <wsa:MessageID>uuid:uuuuvvvv-wwww-xxxx-yyyy-zzzzzzzzzzzz</wsa:MessageID>
        <wsa:RelatesTo>uuid:aaaabbbb-cccc-dddd-eeee-ffffffffffff</wsa:RelatesTo>
    </env:Header>
    <env:Body>
    ...
    </env:Body>
</env:Envelope>
```

U poruci odgovora element ws:RelatesTo sadrži ws:MessageID od poruke zahtjeva koja je pokrenula komunikaciju, dopuštajući da se te dvije poruke međusobno povežu. Način na koji se te informacije šalju u aplikacijski sloj ovisi o sučelju između Web servisa i slojeva aplikacija.

4.4.2. Uzorak duge razmjene poruka

Neki poslovni procesi zahtijevaju složenije razmjene poruke između Web servisa od uzorka zahtjev – odgovor. To može rezultirati razmjenom poruka koje se nastavljaju između dva Web servisa dulje vrijeme. Ti dugotrajni razgovori zahtijevaju dodatne informacije o korelaciji kako bi se osiguralo da se stanje poruke zadrži između servisa koji komuniciraju.

Primjer dugotrajnog razgovora može biti zahtjev za narudžbenicom i postupak njezinog ispunjavanja:

1. Kupac šalje zahtjev za cijenom proizvoda
2. Dobavljač odgovara cijenom proizvoda
3. Kupac sastavlja narudžbu protivno cijenama
4. Dobavljač odgovara potvrdom narudžbe
5. Dobavljač pruža detalje o dostavi informacija
6. Kupac potvrđuje primitak narudžbe

Kao i kod uzorka zahtjev – odgovor, aplikacija ili posredničkom sloju Web servisa može se dodijeliti zadatak da se sve poruke u dugotrajnom razgovoru ispravno upravljaju. Ti slojevi možda trebaju razlikovati mnoge slučajeve dugotrajnog razgovora između dva servisa. Stoga bi bilo korisno upotrijebiti identifikator razgovora u elementu env:Body poslovnog dokumenta ili dizajnirati zasebna zaglavlja koja će sadržavati identifikator razgovora i koja će biti umetnuta u svaku SOAP poruku koja je dio tog razgovora. [5]

4.5. WSDL

Web Services Description Language (WSDL) specifikacija nastala je za opisivanje i objavljivanje formata i protokola Web servisa je standardiziran način. Standardi sučelja Web servisa neophodni su kako bi se osiguralo da se ne mora stupati u interakciju s svakim poslužiteljem na internetu prije početka korištenja Web servisa.

WSDL elementi sadrže opis podataka, obično pomoću jedne ili više XML shema koje se šalju na Web servis tako da i SOAP pošiljatelj i SOAP prijemnik razumiju podatke koji se razmjenjuju. WSDL elementi također sadrže i opis operacija koje se trebaju izvršiti na tim podacima, tako da SOAP prijemnik zna kako ih obraditi podatke, a SOAP pošiljatelj zna kako poslati zahtjev. [6]

Obje strane koje sudjeluju u interakciji preko Web servisa moraju imati pristup istom WSDL-u kako bi se razumjeli. Drugim riječima SOAP pošiljatelj i SOAP prijemnik koju su uključeni u interakciju preko Web servisa moraju imati pristup istoj XML shemi. SOAP pošiljatelj mora znati kako pravilno oblikovati poruku, a SOAP prijemnik treba razumjeti kako na pravilan način interpretirati tu poruku. Sve dok oba dvije strane posjeduju istu WSDL datoteku u pozadini Web servisa može se događati svašta. [6]

Web servisi su u većini slučajeva implementirani pomoću programskih jezika za komunikaciju s internetom kao što su Java servleti koji zovu pozadinski program ili objekt. Te implementacije Web servisa također su obično temeljene na WSDL-u ili su opisane pomoću WSDL-a. To znači da se novi servisi mogu generirati pomoću WSDL-a ili se već postojeći servisi mogu opisati pomoću WSDL-a.

4.5.1. WSDL elementi

WSDL rastavlja Web servise na tri specifična, prepoznatljiva elementa koja se mogu kombinirati ili ponovno koristiti nakon definiranja. Mapiranje iz postojećih aplikacija znači mapiranje tih elemenata koji identificiraju sadržaj i vrste podataka poruka, operacije koje se izvode za određene poruke i specifična spajanja protokola ili vrsti transporta za razmjenu poruka s operacijama preko mreže. Unutar tih elemenata su daljnji podelementi ili dijelovi:

- **Vrste podataka** – u obliku XML shema ili nekog drugog mehanizma. Koristiti će se u SOAP porukama
- **Poruka** – apstraktna definicija podatka, u obliku poruke koja se prikazuje kao cijeli dokument ili kao argumenti koji se prenose za pozivanje metode
- **Operacija** – apstraktna definicija operacije, kao što je nazivanje metode, red poruka ili poslovni proces koji će prihvatiti i obraditi poruku
- **Vrsta porta** – apstraktni skup operacija koje su mapirane na jednu ili više krajnjih točaka, što određuje zbirku operacija za spajanje. Zbirka operacija zbog toga što je apstraktna može se mapirati na više transporta kroz različite veze
- **Spajanje** – konkretni protokol i formati podataka za operacije i poruke definirane za određenu vrstu porta
- **Port** – kombinacija spajanja i mrežne adrese, pružajući ciljnu adresu servisne komunikacije

- **Servis** – kolekcija povezanih krajnjih točaka koje obuhvaćaju definiciju servisa u datoteci. Servisi mapiraju spajanje u port i definiraju bilo kakve definicije proširivosti [6]

Ovi dijelovi Web servisa se obično generiraju pomoću alata koji pretvaraju postojeće meta podatke pozadinske aplikacije u XML shemu te se zatim spajaju u WSDL datoteku. To uvelike olakšava posao programerima jer bi u protivnom oni bili zaduženi za pisanje WSDL datoteke od nule.

4.5.2. Proširivi WSDL okvir

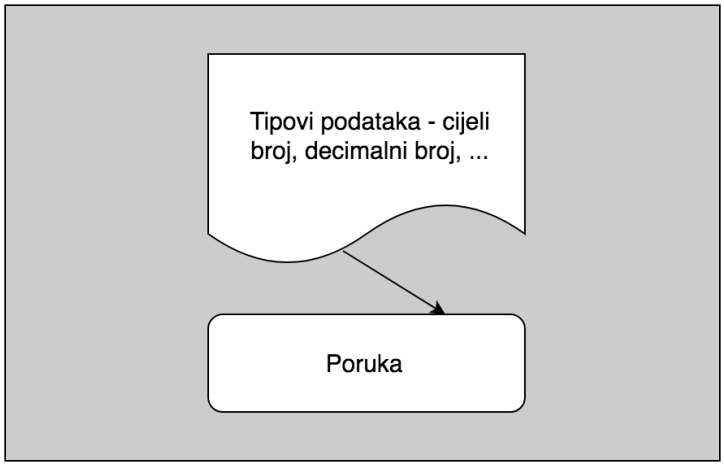
WSDL je proširiv okvir kao i ostale integracije XML okvira. Spajanje za SOAP na primjer proširuje WSDL kao i spajanje za HTTP GET i POST te MIME. Na taj način WSDL odvaja apstraktnu definiciju krajnjih točaka i poruka od njihovih konkretnih mrežnih implementacija i povezivanja podataka, što dopušta ponovnu upotrebu apstraktnih definicija. [6]

WSDL definira tri glavna elementa koja se mogu definirati zasebno, to su tipovi podataka, operacija te spajanja. WSDL se sastoji od sedam dijelova koji su već opisani ali ova tri glavna elementa se mogu razviti kao zasebni dokumenti, te potom kombinirati ili ponovno koristiti za izgradnju novih WSDL datoteka. Glavni elementi su podijeljeni u skladu s njihovom razinom apstrakcije u Web servisima. Tipovi podataka su najzahtjevniji element za definiranje. Oni definiraju tipove podataka koji će se razmjenjivati putem XML poruka u Web servisu. Tipovi podataka se mogu definirati jedanput i biti referencirani od bilo koje operacije Web servisa. Operacije predstavljaju sljedeću razinu apstrakcije određujući prijenos podataka koji se šalju na Web servis ili koje šalje Web servis. Spajanje, zadnja razina apstrakcije, definira vrstu transporta koji se koristi za prosljeđivanje poruke.

4.5.2.1. Definiranje tipova podataka

Na svojoj najapstraktnijoj razini, Web servisi šalju i primaju XML dokumente sa udaljenih softvera. Prvi zadatak u definiraju Web servisa je definiranje tipova podataka na temelju softverskih zahtjeve Web servisa. Web servis treba definirati svoje ulaze i izlaze te kako se oni mapiraju u ulaz i izlaz iz servisa. WSDL tipovi podataka brinu o tome. Tipovi su XML dokument ili dio dokumenta. WSDL omogućuje da se tipovi definiraju u zasebnim elementima tako da se isti tipovi mogu koristiti u više Web servisa. [6]

Vrste podataka rješavaju problem kako definirati tipove podataka i formate koje namjeravate koristiti u svojim Web servisima. Podaci o tipu podataka dijele se između pošiljatelja i prijemnika. Primatelji poruka se trebaju strogo držati podataka pomoću kojih je pošiljatelj kodirao podatke kako bi znali na pravi način dekodirati dobivene podatke. Tipovi podataka se definiraju pomoći XML shema te su potpuno proširivi. To se većinom radi na način da se novi složeniji tipovi podataka definiraju unutar WSDL datoteke. Tipovi podataka se definirati unutar WSDL datoteke ili unutar drugih datoteka navedenih u tom elementu. [6]



Slika 8. Mapiranje tipova podataka u poruku (prema: Newcomer, 2002)

Kako je prikazano na slici 8 jedan ili više tipova podataka preslikavaju se u poruke. Ista poruka se također može mapirati u više operacija. Tipovi podataka su u većini slučajeva bilo koji tipovi podataka koji su podržani pomoću XML sheme kao cijeli brojevi, decimalni brojevi, boolean, datum i drugi. Tipovi podataka također mogu biti i složeniji kao što su strukture, polja i drugi. Tipovi podataka su jednostavnije sheme ili sheme koje definiraju složene tipove podataka. Kao i kod ostalih područja WSDL-a tipovi nisu ograničeni na XML sheme jer nitko ne očekuje da jedna vrsta sustava može biti opisati sve moguće formate poruka za Web servise.

4.5.2.2. Definiranje operacija

Razina apstrakcije koja dolazi nakon tipova podataka su operacije. Operacije se odnose na zahtjeve Web servisa te identifikaciju tipa operacije koja se obavlja za određeni tip poruke ili skupa poruka. Operacije je definirana na način da Web servis zna kako tumačiti podatke i ako je potrebno koje podatke treba vratiti kao odgovor.

Operacije se definiraju u skladu s uobičajenim uzorcima kao što je jednosmjerni ili uzorak zahtjev – odgovor. WSDL ne definira specifične definicije za ostale operacije, no složenije interakcije mogu se konstruirati kombinacijom tim osnovnih vrsta. [6]

Slika 9. Grupiranje tipova poruka u operaciju (prema: Weerawarana, 2005)

Kako je prikazani na slici 9 operacije mogu grupirati poruke koje su ulazne te poruke koje su izlazne kako bi se implementirao uzorak zahtjev – odgovor.

WSDL ima četiri tipa operacija, a to su:

- **Jednosmjerne** – u osnovi se misli na to da se poruka šalje bez da se zahtjeva povratni odgovor
- **Zahtjev – odgovor** – u osnovi pošiljatelj šalje poruku a primatelj šalje odgovarajući odgovor
- **Operacija traženja odgovora** – jednostavan zahtjev koji traži odgovor koji ne sadrži ulazne podatke. To je zahtjev za dobivanjem poruke i ne uključuje slanje poruke, u smislu WSDL poruke koja se sastoji od jedne ili više definiranih vrsta. Ova operacija je obrnuta jednosmjernoj operaciji
- **Obavijest** – ova vrsta operacije definira više prijemnika za poruku, često uključuje mehanizam pretplate na neku određenu informaciju [6]

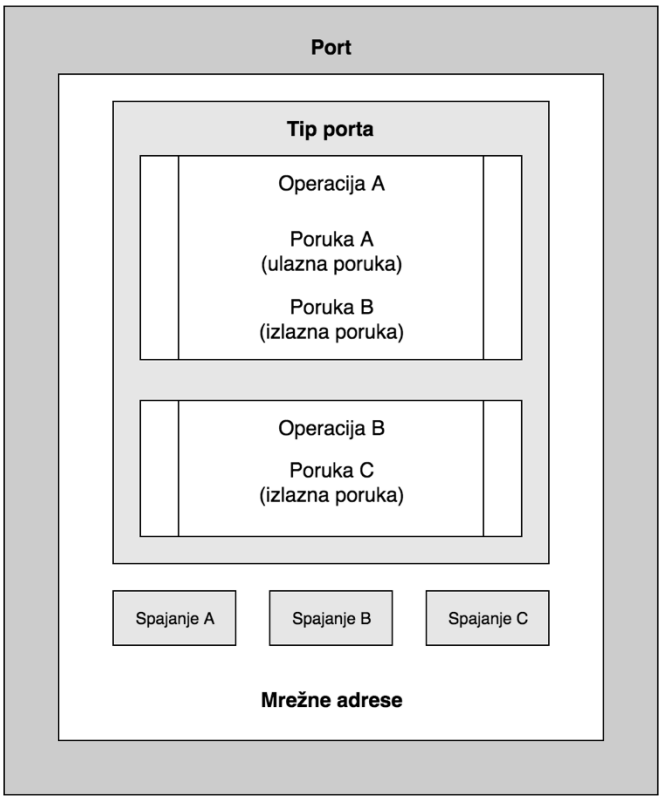
4.5.2.3. Definiranje spajanja

Nakon što se definiraju osnovi tipovi podataka i vrste operacija one moraju biti mapirane na određeni transportni protokol, krajnju točku ili adresu na koju se podaci šalju i na kojima je moguće pronaći i pozvati određene operacije Web servisa. Ovaj korak je vrlo bitan iz razloga jer iste operacije i tipovi podataka mogu biti mapirani na više transportnih protokola i Web servis može potencijalno smješten na više krajnjih točaka.

Prvo se operacije grupiraju u tipove porta. Nakon toga se svaki tip porta u jedan ili više specifičnih porta koji predstavljaju različite vrste transporta preko kojih bi mogao biti dostupan servis. Proširenja spajanja transporata zatim se prenose u svaki pojedini port kako bi definirali informacije potrebne za pružanje određenog servisa preko određene vrste transporta. Proširenja spajanja transporata ispod tipova podataka, tipova operacija i tipova porta identificiraju primatelja podataka i operacije koje treba izvršiti. Dakle za bilo koji Web servis moguće je i potrebno definirati podatke, operacije nad podacima i mjesto na koje se podaci šalju i na koji način. [6]

Tip porta predstavlja logičku grupaciju operacija slično kao klasa u Java programskom jeziku. Ako je operacija analogna metodi u objektu i ako su poruke analogne ulaznim i izlaznim argumentima, tip porta je analogan definiciji objekta koji potencijalno sadrži više metoda. Ali te analogije ne postoje jer je WSDL proširiv i pruža razinu apstrakcije višu od one koju pružaju objektno orijentirani programski jezici. [6]

Port se koristi kako bi se otkrili skupovi operacija ili tipovi porta preko određene vrste transporta. Port se može grupirati u jednu ili više spajanja što označava kako su servisi povezani preko mreže. Port identificira jednu ili više transportnih spajanja za određeni tip porta. [6]

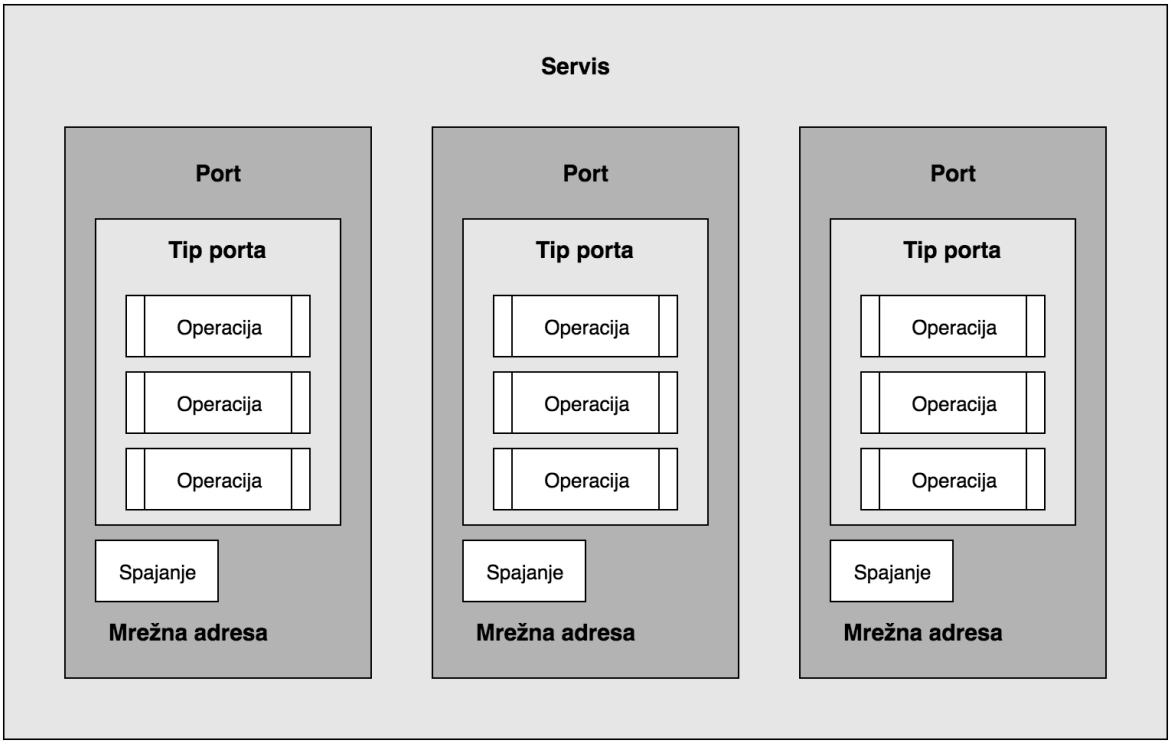


Slika 10. Kombinacija operacija, spajanja i mrežne adrese u WSDL-u (prema: Weerawarana, 2005)

Slika 10 prikazuje koncept porta u WSDL-u, koji zajedno objedinjava operacije, spajanje i definiranje URL-a za specifičnu IP adresu na kojoj može biti prikazana implementacija Web servisa.

Servis dopušta da određena krajnja točka u udaljenoj aplikaciji izabere otkrivanja više kategorija operacija na različite vrste interakcija. Na primjer jedna kategorija može sadržavati skup interakcija usmjerenih na dokument kako bi se asinkrono razmjenjivala i dovršila narudžbenica za buduću pošiljku. Druga kategorija može sadržavati skup RPC orijentiranih interakcija za sinkroniziranje interakcije na nalogu za otpremu. [6]

Kako je prikazano na slici 11 servisni dio WSDL-a obuhvaća jedan ili više ulaza slično načinu na koji klasa objekta može sadržavati više objekata.



Slika 11. Servis kao kolekcija portova (prema: Weerawarana, 2005)

4.5.3. WSDL povezna područja imena

WSDL uključuje područje imena koje je specifično za korištenje unutar određenog dokumenta. Osim vlastito definiranih područja imena, definirani su u specifikaciji je definirano nekoliko područja imena važnih za WSDL:

- **http://schemas.xmlsoap.org/wsdl/** – korišten za WSDL okvir
- **http://schemas.xmlsoap.org/wsdl/soap** – koristi se za definiranje SOAP omotnice u WSDL vezama za SOAP
- **http://schemas.xmlsoap.org/wsdl/http** – koristi se za HTTP GET i POST spajanja u WSDL-u
- **http://schemas.xmlsoap.org/wsdl/mime** – koriste se za HTTP MIME spajanje
- **http://schemas.xmlsoap.org/soap/encoding** - koristi se za shemu SOAP v1.1 enkodiranja
- **http://schemas.xmlsoap.org/soap/envelope** – koristi se za SOAP v.1.1 omotnicu
- **http://www.w3.org/2001/XMLSchema** – koristi se za prostor imena XML shema
- **tns** – prefiks područja imena koji se koristi u konvenciji za označavanje trenutnog dokumenta [6]

4.6. Primjer SOAP Web servisa

Jedan od primjera SOAP Web servisa je Web servis za kontrolu težine korisnika koji je implementiran kao dio praktičnog dijela ovog rada. SOAP Web servis za kontrolu težine korisnika pruža tri metode, metoda za dodavanje nove težine korisnika, metodu za dobivanje zadnje težine korisnika te metodu za dobivanje liste težina za korisnika u određenom vremenskom periodu. Krajnja točka preko koje se može pristupiti svim metodama nalazi se na URI-u http://get-information-services.herokuapp.com/soap.

SOAP poruka zahtjeva za pozivanje metode za dodavanje nove težine korisnika mora biti u XML formatu u imati sljedeću strukturu:

```
<soapenv:Envelope xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
   xmlns:art="http://www.GetInformationsServices.foi.unizg.com/soap">
   <soapenv:Header/>
   <soapenv:Body>
      <art:addWeightRequest>
         <art:username>ivic</art:username>
         <art:weight>81.5</art:weight>
         <art:height>179</art:height>
      </art:addWeightRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP poruka odgovora metode za dodavanje nove težine korisnika dobivena je u XML formatu i ima sljedeću strukturu:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:addWeightResponse
            xmlns:ns2="http://www.GetInformationsServices.foi.unizg.com/soap">
            <ns2:statusCode>OK</ns2:statusCode>
        </ns2:addWeightResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP poruka zahtjeva za pozivanje metode za dobivanje zadnje težine korisnika mora biti u XML formatu u imati sljedeću strukturu:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ar
t="http://www.GetInformationsServices.foi.unizg.com/soap">
   <soapenv:Header/>
   <soapenv:Body>
      <art:getLastWeightByUserRequest>
         <art:username>ivic</art:username>
```

```
      </art:getLastWeightByUserRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP poruka odgovora metode za dobivanje zadnje težine korisnika dobivena je u XML formatu i ima sljedeću strukturu:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:getLastWeightByUserResponse xmlns:ns2="http://www.GetInformationsServices.foi.unizg.com/soap">
            <ns2:statusCode>OK</ns2:statusCode>
            <ns2:weight>
                <ns2:date>2018-09-08Z</ns2:date>
                <ns2:weight>81.5</ns2:weight>
                <ns2:height>179</ns2:height>
                <ns2:bmi>25.436161</ns2:bmi>
            </ns2:weight>
        </ns2:getLastWeightByUserResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP poruka zahtjeva za pozivanje metode za dobivanje liste težina korisnika u određenom vremenskom periodu mora biti u XML formatu u imati sljedeću strukturu:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:art="http://www.GetInformationsServices.foi.unizg.com/soap">
   <soapenv:Header/>
   <soapenv:Body>
      <art:getWeightListRequest>
         <art:username>ivic</art:username>
         <art:from>2018-09-01Z</art:from>
         <art:to>2018-09-10Z</art:to>
      </art:getWeightListRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP poruka odgovora metode za dobivanje liste težina korisnika u određenom vremenskom periodu dobivena je u XML formatu i ima sljedeću strukturu:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:getWeightListResponse xmlns:ns2="http://www.GetInformationsServices.foi.unizg.com/soap">
            <ns2:statusCode>OK</ns2:statusCode>
            <ns2:weight>
                <ns2:date>2018-09-02Z</ns2:date>
                <ns2:weight>95.0</ns2:weight>
                <ns2:height>179</ns2:height>
                <ns2:bmi>29.649513</ns2:bmi>
            </ns2:weight>
```

```
            <ns2:weight>
                <ns2:date>2018-09-02Z</ns2:date>
                <ns2:weight>92.0</ns2:weight>
                <ns2:height>179</ns2:height>
                <ns2:bmi>28.713213</ns2:bmi>
            </ns2:weight>
            <ns2:weight>
                <ns2:date>2018-09-02Z</ns2:date>
                <ns2:weight>90.0</ns2:weight>
                <ns2:height>179</ns2:height>
                <ns2:bmi>28.089012</ns2:bmi>
            </ns2:weight>
            <ns2:weight>
                <ns2:date>2018-09-02Z</ns2:date>
                <ns2:weight>87.0</ns2:weight>
                <ns2:height>179</ns2:height>
                <ns2:bmi>27.152712</ns2:bmi>
            </ns2:weight>
            <ns2:weight>
                <ns2:date>2018-09-08Z</ns2:date>
                <ns2:weight>81.5</ns2:weight>
                <ns2:height>179</ns2:height>
                <ns2:bmi>25.436161</ns2:bmi>
            </ns2:weight>
        </ns2:getWeightListResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5. REST Web servisi

REST je osnovni arhitekturalni stil koji se koristi na Webu. REST određuje skup ograničenja koja osiguravaju da klijenti (korisnici Web servisa i Web preglednici) mogu komunicirati s poslužiteljima na fleksibilan način. Za početak bi bilo dobro da se pojasni osnovna terminologija koja se koristi u REST-u:

- **Poslužitelj** – Davatelj usluga. Pruža servise koje klijenti mogu koristiti
- **Klijent** – Korisnik servisa. Može biti Web preglednik ili neka druga aplikacija
- **Resursi** – Sve informacije mogu biti resursi (objekt, slika, video ili informacije o proizvodu, itd.)
- **Prikazivanje** – Specifičan način na koji je resurs prikazan. Na primjer informacije o proizvodu mogu se prikazati pomoći JSON-a, XML-a, HTML-a, itd. Različiti klijenti mogu zahtijevati različite reprezentacije resursa

Roy Fielding je u svojoj doktorskoj dizertaciji u kojoj je predstavio REST dao šest ograničenja koja vrijede za njih:

- **Klijent – Poslužitelj** – Trebao bi postojati poslužitelj i klijent. To omogućuje labavi spoj i slobodnu nadogradnju servera i klijenta kada iskrsne novi tehnološki zahtjev.
- **Bez stanja** – Komunikacija između klijenta i poslužitelja bi trebala biti bez pamćenja stanja. Poslužitelj ne bi trebao pamtiti stanje klijenta. Klijenti bi morali dati sve potrebne informacije u zahtjevu kako bi ga poslužitelj znao razumjeti i obraditi.
- **Slojeviti sustav** – Između klijenta i poslužitelja može postojati više hijerarhijskih kao što su prolazi, vatrozidi i zastupnici. Slojevi se mogu dodati, mijenjati, preurediti ili ukloniti transparentno kako bi se poboljšala skalabilnost.
- **Privremena memorija** – Odgovori na poslužitelju moraju biti deklarirani kao privremeno spremljeni ili ne privremeno spremljeni. To će omogućiti klijentu ili njegovim posredničkim komponentama da privremeno spreme odgovor i ponovno ga koriste za kasnije zahtjeve. Time se smanjuje opterećenje na poslužitelju i pomaže u poboljšanju performansi.
- **Jednostavno sučelje** – Svaka interakcija između klijenta, poslužitelja i posredničkih komponenata temelji se na ujednačenosti njihovih sučelja. To pojednostavljuje cjelokupnu arhitekturu jer se komponente mogu samostalno

razvijati sve dok implementiraju dobiveni odgovor. Ograničenje jednostavnosti sučelja dodatno je podijeljeno na još četiri pod ograničenja:

- Identifikacija resursa
- Reprezentacija resursa
- Samo opisujuće poruke
- Hipermedija kao pokretač primjene ili HATEOS

- **Kod na zahtjev** – Klijenti mogu proširiti svoje funkcionalnosti preuzimanjem i iskorištavanjem koda na zahtjev. [7]

Aplikacije koje se pridržavaju ovih ograničenja smatraju se REST aplikacijama. Kako se može primijetiti ova ograničenja ne diktiraju tehnologiju u kojima će se koristiti za razvoj aplikacije. Umjesto toga, pridržavanje ovih smjernica i najboljih primjera iz prakse učiniti će aplikaciju skalabilnom, vidljivom, prijenosnom, pouzdanom i sposobnom za daljnja unaprjeđenja. U teoriji, moguće je napraviti REST aplikaciju pomoću bilo koje mrežne infrastrukture ili transportnog protokola. U praksi, REST aplikacije koriste značajke i mogućnosti Web-a i koriste HTTP kao transportni protokol. [7]

Ograničenje jednostavnosti sučelja ključna je značajka koja razlikuje REST aplikacije od ostalih mrežnih aplikacija. Jednostavno sučelje u REST postiže se apstrakcijama poput resursa, prikazivanja, URI-a i HTTP metoda. U sljedećim poglavljima biti će opisane te apstrakcije.

5.1. Razumijevanje resursa

Temelj REST-a je koncept resursa. Resurs je sve čemu se može pristupiti ili s čim se može manipulirati. Primjeri resursa su video zapisi, objave na forumima, korisnički profili, slike, pa čak i stvari ako osobe ili uređaji. Resursi se obično odnose na druge resurse. Na primjeru Web trgovine, korisnik može naručiti bilo koji broj proizvoda. U ovom slučaju, resurs proizvoda povezani su s odgovarajućim resursom narudžbe. Također je moguće da se resurs grupira u kolekcije. Na istom primjeru narudžbe predstavljaju kolekciju resursa narudžba. [7]

5.1.1. Identifikacija resursa

Prije nego što se krene u komunikaciju s resursom i s njegovim korištenjem, potrebno ga je znati identificirati. Web pruža URI za identificiranje resursa. Sintaksa URI-a je

`scheme:scheme-specific-part`. Na primjeru URI-a http://get-information-services.herokuapp.com/rest/forum/category, `http` dio označava shemu, tj. ukazuje na to da HTTP shema treba biti korištena za interpretiranje ostatka URI-a. HTTP shema ukazuje na to da je traženi resurs u primjeru smješten na uređaju s imenom *get-information-services.herokuapp.com*. Tablica u nastavku prikazuje različite primjere URI-a i resurse koje predstavljaju:

Tablica 2. Predstavljanje resursa pomoći URI

URI	Opis resursa
http://get-information-services.herokuapp.com/rest/forum/category	Predstavlja kolekciju objava na forumu
http://get-information-services.herokuapp.com/rest/forum/topic?id=1	Predstavlja objavu na forumu s identifikatorom 1. Takvi se resursi zovu singleton resursi
http://get-information-services.herokuapp.com/rest/forum/1/comments	Predstavlja kolekciju komentara koji su povezani s objavom na blogu koja ima identifikator 1. Kolekcije kao ove koje se nalaze pod nekim drugim resursom nazivaju se pod kolekcije.

(Izvor: Izrada autora)

Iako URI jedinstveno identificira resurs, moguće je da resurs ima više od jednog URI-a. Na primjer Facebook-u se može pristupiti pomoću URI-a http://www.facebook.com ili http://www.fb.com. Pojam URI alias se koristi se za označavanje takvih URI-a koji identificiraju isti resurs. URI aliasi pružaju fleksibilnost i dodatnu pogodnost kao što je upisivanje manje znakova kako bi se pristupilo resursu. [7]

5.1.2. URI predlošci

Prilikom rada sa REST-om pojaviti će se slučajevi kada treba predstavljati strukturu URI-a a ne sam URI. U primjeru aplikacije za forum koja je implementirana u praktičnom dijelu ovog rada, URI oblika http://get-information-services.herokuapp.com/rest/forum/1/comments će

dohvatiti sve komentare na objavu foruma sa identifikacijskim brojem 1. Slično tome URI http://get-information-services.herokuapp.com/rest/forum/2/comments će dohvatiti sve komentare na objavu foruma sa identifikacijskim brojem 2. U ovom primjeru, za korisnike koji koriste ovaj REST bilo bi prikladno da poznaju strukturu http://get-information-services.herokuapp.com/rest/forum/topicId/comments koja označava raspon URI-a, radije nego konkretni URI. Standardizirani URI predložak za ovaj scenarij bi bio http://get-information-services.herokuapp.com/rest/forum/{topicId}/comments.

Vitičaste zagrade ukazuju na to da je godina u predlošku varijabla. Klijenti koji konzumiraju REST mogu uzeti ovaj URI predložak kao unos, zamijeniti varijablu sa pravom vrijednosti i dohvatiti odgovarajuće resurse. Na strani poslužitelja, URI predlošci dozvoljavaju poslužiteljskom kodu analizu i dohvaćanje vrijednosti varijable ili odabranih dijelova URI-a. [7]

5.2. Prikaz resursa

REST resursi su apstraktni entiteti. Podatci i meta podaci koji čine REST resurse moraju biti serijalizirani u određenu vrstu prikaza prije nego što se šalju korisniku. Ovaj prikaz resursa se može smatrati kao snimka stanja resursa u određenom vremenskom trenutku. Kada uzmemo primjer Web trgovine, njezina baza podataka pohranjuje informacije o svim dostupnim proizvodima. Kada online kupac koristi svoj Web preglednik za kupnju proizvoda i zatraži njegove pojedinosti, aplikacija će dati detalje o proizvodu kao Web stranicu u HTML-u. Drugi primjer je kada programer implementira mobilnu aplikaciju te su mu potrebni podaci o proizvodu, REST aplikacija može vratiti te podatke u obliku XML ili JSON formatu. U oba slučaja korisnici nisu stupili u interakciju s stvarnim resursom već s njegovim prikazom. [7]

Kao što je navedeno u primjeru proizvoda, isti resurs može imati više vrsta prikaza. Ti prikazi se mogu razlikovati od tekstualnim HTML, XML, JSON formata pa do binarnih formata kao što su PDF, JPEG, MP4, itd.

Moguće je da korisnik zahtjeva određenu vrstu zapisa, te se taj proces zove pregovaranje o sadržaju. Postoje dvije moguće strategije pregovaranja o sadržaju:

- Dodavanje fiksnih dijelova na kraj URI-a. U ovoj strategiji korisnik koji traži pojedinosti o proizvodu u JSON formatu koristiti će URI

http://example.com/products/12.json. Drugi korisnik koji traži pojedinosti o proizvodu u XML formatu će koristiti URI http://example.com/products/12.xml.

- Korištenjem svojstva prima *(eng. accept)* u zaglavlju zahtjeva. Korisnici mogu popuniti svojstvo prima sa željenim prikazom i poslati ga zajedno s zahtjevom poslužitelju. Aplikacija koja upravlja resursom koristi vrijednost koja je specificirana u zaglavlju kako bi se koristio odgovarajući prikaz za odgovor. Specifikacija RFC 26163 pruža detaljni skup pravila za određivanje jednog ili više formata i njihovog prioriteta. [7]

5.3. HTTP metode zahtjeva

Ograničenje *Jednostavno sučelje* ograničava interakciju korisnika i poslužitelja kroz niz standardiziranih operacija. Na Webu, HTTP pruža devet metoda koje klijentima omogućuju interakciju i manipulaciju resursima. Neke od najčešće korištenih metoda su GET, POST, PUT i DELETE.

5.3.1. GET metoda

GET metoda se koristi za preuzimanje reprezentacije resursa. Na primjer, URI http://example/posts/1 vraća podatke o objavi koja ima identifikator 1. Drugi primjer je URI http://example/posts koji vraća listu svih objava na blogu. Budući da GET metoda ne mijenja stanje na poslužitelju, ona se smatra sigurnom metodom.

GET zahtjev na URI http://blog.example/posts/1:

```
GET      /posts/1 HTTP/1.1
Accept:  application/json
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Host: example.com
```

Odgovor:

```
Content-Type: application/json; charset=UTF-8
Date: Sat, 11 Aug 2018 20:00:00 GMT
Server: Apache

BODY

{"naslov": "Prva objava", "objava": "Sadržaj prve objave!"}
```

5.3.2. POST metoda

POST metoda se koristi za stvaranje resursa. Tipično se koristi za stvaranje resursa u pojedinačnim zbirkama resursa. Na primjer POST metoda se može koristiti za kreiranje nove objave u blog aplikaciji. Metoda POST se ne smatra sigurnom metodom jer mijenja stanje resursa.

POST zahtjev ne mora znati URI resursa. Poslužitelj je odgovoran za dodjeljivanje ID resursa i odlučivanje o URI-u na kojemu će resurs živjeti. Zaglavlje odgovora sadrži svojstvo lokacija *(eng. location)* koji ima URI do novo kreiranog resursa. [7]

POST zahtjev:

```
1.  POST /posts      HTTP/1.1
2.
3.  Accept: */*
4.  Content-Type: application/json
5.  Content-Length: 63
6.  Host: example.com
7.
8.  BODY
9.
10. {"naslov": "Druga objava", "objava": "Još jedna objava!"}
```

Odgovor:

```
1.  Content-Type: application/json
2.  Location: posts/12345
3.  Server: Apache
```

5.3.3. PUT metoda

PUT metoda omogućuje klijentu izmjenu stanja resursa. Klijent mijenja stanje resursa i šalje ažuriranu reprezentaciju poslužitelju pomoću PUT metode. Po primitku zahtjeva, poslužitelj zamjenjuje stanje resursa novim stanjem. Metoda PUT se ne smatra sigurnom metodom jer mijenja stanje resursa.

U sljedećem primjeru šalje se PUT zahtjev za ažuriranje objave s identifikatorom 1. Zahtjev sadrži ažuriranu objavu zajedno sa svim ostalim poljima koje čine kompletnu objavu kao što je naslov. Poslužitelj će nakon uspješne obrade vratiti status kod 200, što znači da je zahtjev uspješno obrađen.

```
1. PUT /posts/1     HTTP/1.1
2.
3. Accept: */*
4. Content-Type: application/json
5. Content-Length: 65
6. Host: example.com
7.
8. BODY
9.
10. {"naslov": "Prva objava", "objava": "Uređena prva objava!"}
```

HTTP semantika diktira da kao dio PUT zahtjeva šalje puna reprezentaciju resursa, koja uključuje ažurirani naslov kao i druge atribute kao što je tijelo objave i drugi atributi koji se nisu promijenili. Međutim ovaj pristup zahtjeva da klijent ima potpuni prikaz resursa, što možda neće biti moguće ako je resurs velik ili ima puno veza. Da bi se podržalo djelomično ažuriranje dodane je metoda PATCH. [7]

5.3.4. DELETE metoda

DELETE metoda kako i samo ime govori, traži resurs koji treba obrisati. Po primitku zahtjeva poslužitelj briše resurs. Ovisno o implementaciji sustava, resurs se možda neće fizički izbrisati, nego samo prestati prikazivati. Metoda DELETE se ne smatra sigurnom metodom jer mijenja stanje resursa.

Nakon uspješnog brisanja resursa, budući GET zahtjevi na tom resursu dobili bi pogrešku putem HTTP status koda 404 (Nije pronađen). Za resurse koje treba dugo kako bi se obrisali poslužitelj obično šalje potvrdu da je primio zahtjev i da radi na njemu. [7]

U sljedećem primjeru klijent zahtjeva da se izbriše objava s identifikatorom 1. Po završetku obrade poslužitelj može vratiti HTTP status kod 200 (OK) ili 204 (bez sadržaja), što znači da je zahtjev uspješno obrađen.

```
1. Delete /posts/1  HTTP/1.1
2. Content-Length: 0
3. Content-Type: application/json
4. Host: example.com
```

5.3.5. HEAD metoda

Ponekad bi možda klijent htio provjeriti postoji ili određeni resurs bez da ima potrebu za njegovom reprezentacijom. U nekom drugom slučaju klijent bi htio prvo provjeriti da li određeni resurs postoji pa ovisno o tome krene u preuzimanje njegove reprezentacije. Za ove navedene

operacije HEAD metoda je prikladna za to. Kao i GET metoda, metoda HEAD se smatra sigurnom metodom jer ne mijenja stanje resursa.

HEAD metoda omogućuje klijentu samo dohvaćanje meta podataka povezanih s resursom. Ni jedna reprezentacija resursa nije poslana klijentu. Ovi meta podaci prikazani u HTTP zaglavlju biti će identični informacijama poslanima u GET zahtjevu. Korisnik koristi ove meta podatke kako bi utvrdio dostupnost resursa i nedavne njegove izmjene. [7]

Primjer HEAD zahtjeva:

```
1. HEAD      /posts/1 HTTP/1.1
2. Accept:  application/json
3. Accept-Encoding: gzip, deflate
4. Accept-Language: en-US,en;q=0.5
5. Connection: keep-alive
6. Host: example.com
```

Odgovor:

```
1. Connection:  Keep-Alive
2. Content-Type: application/json; charset=UTF-8
3. Date: Sat, 11 Aug 2018 20:00:00 GMT
4. Server: Apache
```

5.3.6. PATCH metoda

Kao što je prije navedeno, HTTP specifikacija zahtjeva da korisnik pošalje čitavu reprezentaciju resursa kad koristi metodu PUT. Metoda PATCH koristi se za obavljanje djelomično ažuriranje resursa. Metoda PATCH se ne smatra sigurnom metodom jer mijenja stanje resursa.

Primjer uporabe PATCH metode za ažuriranje naslova objave:

```
1. PATCH /posts/1   HTTP/1.1
2.
3. Accept: */*
4. Content-Type: application/json
5. Content-Length: 59
6. Host: example.com
7.
8. BODY
9.
10. {"zamijeni": "naslov", "sadržaj": "Uređeni naslov"}
```

Tijelo zahtjeva sadržava opis promjena koje je potrebno obaviti nad resursom. U primjeru tijelo zahtjeva ima svojstvo zamijeni kako bi se označilo da vrijednost polja naslov treba ažurirati.

Ne postoji standardizirani format za opisivanje promjena na poslužitelju kao dio PATCH zahtjeva. Različita implementacija može koristiti različite formate. [7]

5.3.7. OPTIONS metoda

Ponekad je korisno jednostavno prepoznati mogućnosti Web-servisa s kojim se želi komunicirati prije nego što se napravi konkretni zahtjev. U tu svrhu HTTP pruža metodu OPTIONS. U nastavku je prikazan OPTIONS zahtjev:

```
1. OPTIONS * HTTP/1.1
2. Host: 127.0.0.1
```

Znak zvjezdice nije valjani resurs ali omogućuje OPTIONS zahtjevu da se raspita o sposobnostima poslužitelja bez konteksta bilo kojeg specifičnog resursa. Odgovor na OPTIONS zahtjev će detaljno opisati mogućnosti koje se mogu osigurati za bilo koji resurs koji Web servis. Svojstvo dopušta *(eng. allow)* u zaglavlju odgovora prikazuje načine zahtjeva koji su podržani. [8]

```
1. HTTP/1.1 200 OK
2. Date: Sat, 11 Aug 2018 20:00:00 GMT
3. Server: Apache/1.3.22 (Unix)  (Red-Hat/Linux) mod_python/2.7.8 Python/1.5.2
4.          mod_ssl/2.8.5 OpenSSL/0.9.6b DAV/1.0.2 PHP/4.0.6 mod_perl/1.26
5.          mod_throttle/3.1.2
6. Content-Length: 0
7. Allow: GET, HEAD, OPTIONS, TRACE
8. Connection: close
```

5.3.8. TRACE metoda

TRACE metoda je jedna od dijagnostičkih metoda. Ova metoda omogućuje klijentu da dobije više perspektiva o zastupnicima koji leže između klijenta i poslužitelja. Budući da svaki zastupnik prosljeđuje TRACE zahtjev na rutu do odredišnog Web poslužitelja, dodaje se u zaglavlje svojstvo preko *(eng. via)*, pri čemu je zastupnik odgovoran za dodavanje zaglavlja. Kada je odgovor poznat, sadržaj je zapravo konačni zahtjev, uključujući svojstvo preko. [8]

5.3.9. CONNECT metoda

Metoda CONNECT rezervirana je za korištenje posrednih poslužitelja za stvaranje tunela odredišnom poslužitelju. Posrednik, a ne HTTP klijent šalje CONNECT zahtjev na odredišni poslužitelj. [8]

Tunel je drugačiji od zastupnika zbog toga jer ne interpretira HTTP zahtjeve u privremenu memoriju. Iz perspektive klijenta i poslužitelja tunel je transparentan. Tunel ostaje uspostavljen sve dok TCP veza ostaje otvorena. Veza je zatvorena nakon što odredišni poslužitelj zatvori vezu s klijentom. [8]

Najčešća upotreba CONNECT metode je kada Web klijent mora koristiti zastupnika kako da bi zatražio siguran resurs pomoću SSL-a ili TLS-a. Klijent će tunelirati zahtjev putem zastupnika tako da zastupnik jednostavno usmjerava HTTP poruke od Web poslužitelja bez pokušaja da ih pregleda ili prevodi.

5.4. HTTP status kodovi

HTTP status kodovi omogućuju poslužitelju da prenese rezultate obrade zahtjeva klijentu. Status kodovi grupiraju se u sljedeće kategorije:

- **Informativni kodovi** – Status kodovi koji ukazuju da je poslužitelj primio zahtjev ali ga nije obradio. Ovi status kodovi se nalaze u grupi 100
- **Kodovi uspješnosti** – Status kodovi koji ukazuju da je zahtjev uspješno primljen i obrađen. Ovi status kodovi se nalaze u grupi 200
- **Kodovi preusmjeravanja** – Status kodovi koji ukazuju na to da je zahtjev obrađen, no klijent mora izvršiti dodatnu radnju za dovršetak zahtjeva. Ova radnja obično uključuje preusmjeravanje na drugu lokaciju kako bi dobili resurs. Ovi status kodovi se nalaze u grupi 300
- **Klijentski kodovi pogrešaka** – Status kodovi koji ukazuju na pogrešku ili problem s korisničkim zahtjevom. Ovi status kodovi se nalaze u grupi 400.
- **Poslužiteljski kodovi pogrešaka** – Status kodovi koji ukazuju na pogrešku na poslužitelju tijekom obrade korisničkog zahtjeva. Ovi status kodovi se nalaze u grupi 500 [7]

HTTP status kodovi imaju veliku ulogu u dizajniranju REST Web servisa jer smisleni status kodovi omogućuju klijentu da reagira na odgovarajući način. Sljedeća tablica prikazuje neke tipične status kodove koji se obično koriste.

Tablica 3. HTTP status kodovi

Status kod	**Objašnjenje**
100 (Nastavak)	Pokazuje da je poslužitelj primio prvi dio zahtjeva, a ostatak zahtjeva bi trebao biti poslan.
200 (OK)	Označava da je sve dobro prošlo s obradom zahtjeva.
201 (Kreiran)	Označava da je zahtjev obrađen i da je kreiran novi resurs.
202 (Primljen)	Označava da je zahtjev prihvaćen ali se još uvijek obrađuje.
204 (Nema sadržaja)	Označava da je zahtjev obrađen i nema rezultata koje bi se poslalo korisniku.
301 (Trajno premješten)	Označava da je traženi resurs premješten na novu lokaciju, a za pristup resursu treba koristiti novi URI.
400 (Loš zahtjev)	Označava da je zahtjev neispravan i da poslužitelj nije u mogućnosti razumjeti zahtjev.
401 (Neovlašten)	Označava da se korisnik treba autenticirati prije pristupanja resursu. Ako zahtjev već sadrži korisnikove podatke za autentikaciju, tada 401 znači da su podaci za autentikaciju pogrešni (npr. pogrešna lozinka).

403 (Zabranjen)	Označava da je poslužitelj razumio zahtjev ali ga odbija ispuniti. To može biti jer se resurs traži s IP adrese koja ja na crnoj listi.
404 (Nije pronađen)	Označava da resurs na traženom URI-u ne postoji.
406 (Neprihvatljiv)	Označava da je poslužitelj sposoban za obradu zahtjeva ali generalni odgovor možda nije prihvatljiv korisniku. To se događa kada klijent postaje previše izbirljiv s prihvaćanjem zaglavlja.
500 (Interna pogreška poslužitelja)	Označava da je došlo do pogreške na poslužitelju prilikom obrade zahtjeva i da se zahtjev ne može dovršiti.
503 (Nedostupan servis)	Prikazuje da se zahtjev ne može dovršiti jer je poslužitelj preopterećen ili se nalazi u stanju redovitog održavanja.

(prema: Varanasi and Belida, 2015)

5.5. WADL

Web Application Description Language (u daljnjem tekstu WADL) je generalno u REST-u što i WSDL u SOAP-u. U načelu WADL je sličan WSDL-u, ali je struktura jezika mnogo drugačija. Dok WSDL definira popis poruka i operacija ili konzumiranje i produciranje nekih od njih, WADL naglašava hijerarhijsku prirodu REST Web servisa. [9]

U REST-u je sve bazirano na resursima. Svaki resurs je predstavljen kao URI. Svaki resurs može definirati i CRUD operacije i ugniježđene resurse. Jednostavan primjer bi bio forum koji je implementiran u praktičnom dijelu ovog rada gdje http://get-information-services.herokuapp.com/rest/forum/category predstavlja listu kategorija na forumu. Sa GET metodom moguće je dobiti listu kategorija, dok je s POST metodom omogućeno dodavanje nove kategorije. Isto tako pomoću GET metode može se pristupiti resursu http://get-

inforamtion-services.herokuapp.com/rest/forum/topic?id=2 koji će dati podatke objave na forumu s identifikacijskim brojem 2. Pomoću REST-a nema niti ograničenja na razinu ugnježđivanja, tako npr. resurs na URI-u http://get-inforamtion-services.herokuapp.com/rest/forum/2/comments daje sve komentare objave na forumu s identifikacijskim brojem 2.

WADL ima za ulogu formalno i precizno opisati raspoložive resurse, metode, zahtjeve i odgovore, baš kao i WSDL. WADL je jedna od opcija za opisivanje dostupnih URI-a u REST-u, iako bi dobro napisan REST morao biti samo opisujući. [9]

U nastavku je naveden jednostavan, prazan WADL dokument:

```
1.  <application xmlns="http://wadl.dev.java.net/2009/02">
2.     <resources base="http://get-information-services.herokuapp.com/rest/forum"/>
3.  </application>
```

U ovom jednostavnom primjeru prikazano je da element <resource> definira baznu adresu za REST. Svi navedeni resursi, koje će biti dodani u nastavku odnose se na ovu adresu. Također može de definirati i više od jednog <resource> elementa, za bolje opisivanje REST Web servisa što je prikazano u sljedećem primjeru:

```
1.  <application xmlns="http://wadl.dev.java.net/2009/02">
2.     <resources base="http://get-information-services.herokuapp.com/rest/forum">
3.        <resource path="category">
4.           <method name="GET" />
5.           <method name="POST" />
6.        </resource>
7.     </resources>
8.  </application>
```

U ovom primjeru definiran je resurs na URI-u http://get-information-services.herokuapp.com/rest/forum/category sa njegovim dvjema metodama. GET metoda za dohvaćanje liste kategorija foruma i POST metoda za dodavanje nove kategorije foruma. Ovisno o zahtjevima, možda će biti dopuštena metoda DELETE za brisanje svih kategorija, te se u tom slučaju i ona definira u WADL dokumentu.

Na početku odlomka dan je primjer URI-a http://get-information-services.herokuapp.com/rest/forum/2/comments koji daje sve komentare objave na forumu s identifikacijskim brojem 2. Identifikacijski broj 2 je samo bio primjer i naravno da se u WADL

dokumentu neće navoditi sve objave. Umjesto toga postoji praktična sintaksa koja je navedena u sljedećem primjeru:

```
<application xmlns="http://wadl.dev.java.net/2009/02">
   <resources base="http://get-information-services.herokuapp.com/rest/forum">
      <resource path="category">
         <method name="GET" />
         <method name="POST" />
      </resource>
      <resource path="{topicId}">
         <param required="true" style="template" name="topicId" />
         <resource path="comments">
            <method name="GET" />
         </resource>
      </resource>
   </resources>
</application>
```

Zamjensko mjesto {topicId} upotrijebljeno je umjesto ugniježđenog resursa. Ovo zamjensko mjesto dokumentira se pomoću elementa <param>. U sljedećem primjeru će biti prikazano kako se element <param> može koristiti u kombinaciji s metodama.

```
<application xmlns="http://wadl.dev.java.net/2009/02">
   <resources base="http://get-information-services.herokuapp.com/rest/forum">
      <resource path="category">
         <method name="GET" />
         <method name="POST" />
      </resource>
      <resource path="{topicId}">
         <param required="true" style="template" name="topicId" />
         <resource path="comments">
            <method name="GET" />
         </resource>
      </resource>
      <resource path="topic">
         <request>
            <param name="id" required="false" default="1" style="query" />
         </request>
      </resource>
   </resources>
</application>
```

Kao što je prethodno prikazano u mogu se samostalno opisati parametri svake metode. U primjeru su definirani parametri upita. To klijentu daje dodatna znanja o prihvatljivim parametrima upita. Kako je prikazano u prethodnom primjeru zahtjev oblika http://get-inforation-services.herokuapp.com/rest/forum/topic?id=2 je važeći identifikator resursa i vraća informacije o objavi na forumu pod identifikacijskim brojem 2 .

Kako u je WSDL-u moguće dokumentirati odgovor Web servisa, tako je i pomoću WADL-a moguće definirati odgovore pojedinih metoda. Za to opisivanje koristi se element <response>

koji je ugniježđen u element <method>. Zahtjev i odgovor mogu definirati točnu gramatiku koju mora slijediti zahtjev ili odgovor. Element <response> također može dokumentirati moguće HTTP status kodove odgovora. Iz svega navedenog može se zaključiti da je WADL agilan i omogućuje da se definiraju samo informacije koje su bitne. [9]

5.6. Primjer REST Web servisa

U nastavku će biti prikazan REST Web servis koji je implementiran kao dio praktičnog dijela ovog rada. REST Web servis ima za ulogu upravljanje resursom korisnika aplikacije. Metode koje se mogu obaviti nad resursom korisnika su dobivanje podatka korisnika, dodavanje novog korisnika te promjena korisničkih podataka i dostupne su pomoću različitih HTTP metoda zahtjeva na URI-u http://get-information-services.herokuapp.com/rest/user.

Metodi koja ima za ulogu pružanje korisničkih podataka za određenog korisnika može se pristupiti pomoću HTTP GET zahtjeva na prethodno naveden URI zajedno sa GET parametrom *username* koji predstavlja korisničko ime korisnika čiji se podaci traže. Rezultat REST Web servisa dobiven je u JSON formatu zapisa i njegova struktura izgleda kao sljedeći primjer.

```
1.  {
2.      "status": "OK",
3.      "response": {
4.          "id": 4,
5.          "username": "peroPeric",
6.          "firstName": "Pero",
7.          "lastName": "Perić",
8.          "email": "pero_peric@foi.hr"
9.      },
10.     "error": ""
11. }
```

Metodi koja ima za ulogu dodavanje novog korisnika u bazu podataka može se pristupiti putem HTTP POST zahtjeva na ranije naveden URI zajedno sa tijelom zahtjeva koje je u JSON formatu zapisa i ima sljedeću strukturu:

```
1.  {
2.    "firstName": "Pero",
3.    "lastName": "Perić",
4.    "username": "peroPeric",
5.    "email": "pero_peric@foi.hr",
6.    "password": "caxcQXxaxihmsnav"
7.  }
```

Rezultat REST Web servisa dobiven je u JSON formatu i ima sljedeću strukturu:

```
{
  "status": "OK",
  "response": "",
  "error": ""
}
```

Metodi koja ima za ulogu promjenu korisničkih podataka može se pristupiti putem HTTP PATCH zahtjeva na ranije naveden URI zajedno za tijelom zahtjeva koje je u JSON formatu i ima sljedeću struktu:

```
{
  "id": 4,
  "username": "peric",
  "firstName": "Pero",
  "lastName": "Perić",
  "email": "pero.peric@foi.hr",
  "password": "caivscSAvsc"
}
```

Rezultat REST Web servisa dobiven je u JSON formatu i ima sljedeću strukturu:

```
{
  "status": "OK",
  "response": "",
  "error": ""
}
```

6. Formati zapisa Web servisa

U ovom trenutku kada se zna što su to Web servisi, te su pobliže objašnjenje vrste Web servisa, vrijeme je da se pobliže objasne i formati zapisa s kojima oni rade. Kao što je navedeno u prijašnjim poglavljima SOAP Web servisi se baziraju na XML formatu zapisa. S druge strane REST Web servisi imaju široku paleta formata zapisa koje mogu koristiti. Tako REST Web servisi mogu raditi s XML, JSON, HTML, običnim tekstom, te binarnim formatima kao što su PDF, JPEG i drugi. Kako su u današnje vrijeme najviše korišteni formati zapisa XML i JSON u nastavku će oni biti detaljnije objašnjeni.

6.1. XML format zapisa

XML je kratica za Extensible Markup Language, pomoću kojega se opisuje dokument. XML se sastoji od mnogo dijelova koji nemaju definiranu strukturu osim one koju im dizajneri sami dodaju.

U XML-u postoje dva glavna dijela, elementi i atributi. Elementi, koji se ponekad nazivaju i čvorovi, su okosnica XML-a i pomoću njih se opisuje struktura XML-a. Zanimljivo kod elementa je to da oni mogu biti sve što je potrebno. [10]

U nastavku je naveden primjer XML dokumenta:

```
1. <?xml version="1.0" ?>
2. <student>
3.    <ime>Pero</ime>
4.    <prezime>Perić</prezime>
5.    <fakultet>FOI</fakultet>
6. </student>
```

U prethodnom primjeru jasno je što predstavljaju podaci, studenta sa imenom, prezimenom i fakultetom. U XML-u postoje otvarajući i zatvarajući elementi koji su neophodni za dobro strukturiranje dokumenta. Svaki element će biti omotan šiljastim zagradama, kao <student>. Element zatvaranja ima isto ime kao i otvarajući element samo što mu se dodaje / na početak, kao </student>. Još je jedino pitanje ostalo što znači prva linija u prethodnom primjeru. To je XML deklaracija i ona je obavezna kako bi alati za parsiranje mogli identificirati u kojem slučaju je dokument XML.

Nakon elementa drugi najčešći dio XML-a je atribut. Atribut se pojavljuje unutar elementa i pomaže u opisivanju tog elementa. Atribut će pomoći opisati element dajući mu dodatni kontekst. [10]

U sljedećem primjeru biti će prikazan primjer s studentom pomoću jednog elementa s više atributa:

```
1.  <?xml version="1.0" ?>
2.  <student ime="Pero" prezime="Perić" fakultet="FOI " />
```

U kojem slučaju koristiti elemente a u kojem slučaju atribute? To uglavnom ovisi o dizajneru dokumenta, ali ako se držimo konvencije tada kada imamo opisne informacije o elementu one bi trebale biti atributi. Isto tako, informacije koje su dio podataka trebale bi biti element. [10]

U gore navedenim primjerima ima smisla da ime, prezime i fakultet stavimo kao atribute a ne kao pod elemente. To je tako iz razloga jer su ta tri dijela izravno vezana za studenta. Ako bi željeli dodati predmete studenta, njih bi bilo pametno staviti kao pod elemente od elementa studenta.

6.2. JSON format zapisa

JSON je kratica za JavaScript Object Notation. JSON je format zapisa neovisan o jeziku i služi za serijalizaciju podataka. JSON ima svoju gramatiku i vrlo je jednostavna. JSON se bazira na sljedećim simbolima:

- Šest strukturnih znakova ***{*** , ***}*** , ***[*** , ***]*** , ***:*** i ***,***
- Nizovi
- Brojevi
- Tri podatka ***true***, ***false*** i ***null*** [11]

Objekti počinju i završavaju s vitičastim zagradama, tj. {objekt}. Polja počinju u završavaju s uglatim zagradama, tj. [polje]. Dvotočka se koristi za odjeljivanje imena i vrijednosti. Višestruki objekti ili polja se odvajaju zarezom. Pomoću ovih jednostavnih pravila mogu se stvoriti bilokakve složenije strukture. JSON podatke si možemo zamisliti kao podatke strukturirane kao objekte s parovima ključ-vrijednost, ili kao sortirana lista vrijednosti, tj. polje. Većina programskih jezika koristi objekte te polja objekata. Jednostavna struktura JSON-a omogućuje mu da bude neovisan o programskom jeziku u kojem se koristi. [11]

```
1. { "ime": "Pero",  "prezime": "Perić",  "fakultet": "FOI" }
```

Podaci su u prethodnom primjeru zatvoreni unutar vitičastih zagrada što označava da se radi o JSON objektu, a ne JSON polju. Ovaj JSON objekt sadrži tri para podataka ključ-vrijednsot koji su razdvojeni zarezom. U ovom primjeru sve vrijednosti su unutar navodnika tako da se mogu smatrati nizovima.

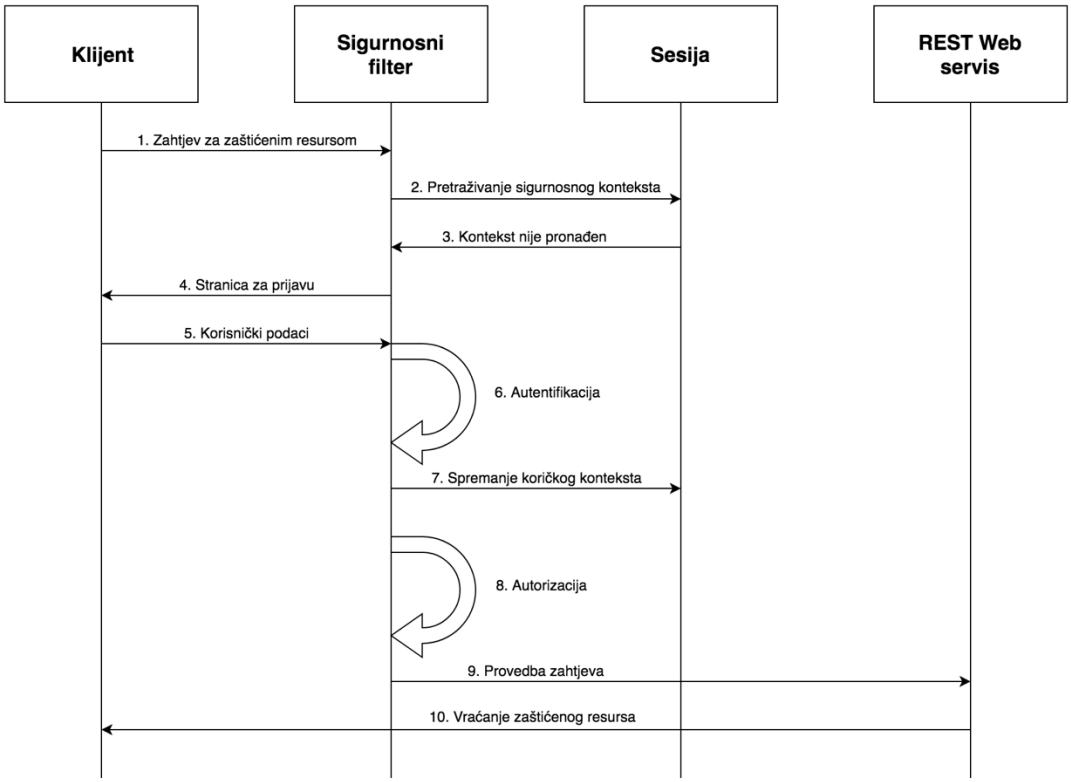
7. Sigurnost Web servisa

Tradicionalne Web aplikacije temelje svoju sigurnost na korisničkom imenu i lozinki s kojima se korisnik identificira. REST Web servisi donose zanimljive sigurnosne probleme jer ih mogu konzumirati različiti klijenti, kao što su Web preglednici i mobilni uređaji. Također ih mogu koristiti i drugi Web servisi, a ta komunikacija u većini slučajeva automatska i nema ljudske interakcije. Također nije neobično da klijenti koriste REST Web servise u ime korisnika. U ovom poglavlju biti će prikazani različiti pristupi autentifikacije i autorizacije koji se mogu koristiti tijekom rada s REST Web servisima. U nastavku će biti objašnjeno šest popularnih metoda koje se koriste za osiguravanje REST Web servisa, a to su:

- Sigurnost temeljena na sesiji
- Osnovna HTTP autentifikacija
- Potvrda autentičnosti
- Sigurnost temeljna na certifikatu
- XAuth
- OAuth

7.1. Sigurnost temeljna na sesiji

Ovaj sigurnosni model oslanja se na sesiju na strani poslužitelja kako bi se zadržao identitet korisnika u svim zahtjevima. U klasičnoj Web aplikaciji kada korisnik želi pristupiti zaštićenom resursu, aplikacija ga preusmjerava na stranicu za prijavu. Nakon uspješne provjere autentičnosti poslužitelj pohranjuje podatke o prijavljenom korisniku u HTTP sesiju. U naknadnim zahtjevima provjera autentičnosti se provodi na temelju podataka koji su spremljeni u sesiji i provjerava se da li korisnik ima pravo pristupa određenom resursu. Ako korisnik nema prava pristupa određenom resursu njegov zahtjev će biti odbijen. Sljedeća slika prikazuje ovaj postupak.



Slika 12. Postupak provođenja modela sigurnosti temeljene na sesiji (prema: Varanasi i Belida, 2015)

Programski okviri kao što je Spring pružaju sve potrebne dijelove pomoću kojih se može implementirati ovaj sigurnosni model. Ovaj pristup je vrlo korišten u slučajevima kada se dodaje novi REST Web servis na već postojeću Web aplikaciju. REST Web servis će uzeti korisničke podatke iz sesije za provjeru prava korisnika i pružanje resursa u skladu s tim. Međutim ovaj pristup krši pravilo REST-a koje govori kako bi REST morao biti bez stanja. Isto tako zbog toga što poslužitelj sprema stanje klijenta, taj pristup nije stabilan. U idealnom slučaju klijent bi trebao čuvati stanje, a poslužitelj bi trebao biti bez stanja. [7]

7.2. Osnovna HTTP autentifikacija

Korištenje obrasca putem kojega se obavlja autentifikacija moguća je samo u slučaju kada korisnik sudjeluje u interakciji s sustavom. Međutim to neće biti moguće kada će Web servisi biti u interakciji s drugim Web servisima.

Osnovna HTTP autentifikacija pruža mehanizam koji klijentima omogućuje slanje podataka o autentičnosti pomoću interaktivnih i neinteraktivnih metoda. U tom pristupu, kada klijent pošalje zahtjev na zaštićeni resurs, poslužitelj šalje odgovor s status kodom 401 i zaglavlje s svojstvom „WWW-Authenticate“. „Basic“ kao vrijednost u zaglavlju označava da će se koristiti osnovna HTTP autentifikacija, a „realm“ svojstvo označava zaštićeno područje na poslužitelju. [7]

U nastavku je naveden primjer ovakvog zaglavlja:

```
1. GET /protected_resource
2. 401 Unauthorized
3. WWW-Authenticate: Basic realm="Example Realm"
```

Po primitku odgovora koji je naveden u prethodnom primjeru klijent spoji korisničko ime i lozinku sa zarezom te pomoću Base64 metode kodira spojeni niz. Zatim ih šalje poslužitelju pomoću standardnog zaglavlja za autorizaciju kako je prikazano u sljedećem primjeru.

```
1. GET /protected_resource
2. Authorization: Basic bHxpY26U5lkjfdk
```

Poslužitelj nakon toga dekodira dobivene podatke te ih provjerava. Nakon uspješne provjere, poslužitelj dovršava zahtjev.

Budući da klijent sadrži podatke o autentičnosti u svakom zahtjevu, poslužitelj postaje bez stanja. Važno je za napomenuti da klijent jednostavno kodira podatke, ali ih ne šifrira. Dakle na SSL/TLS vezama moguće je provesti napad i ukrasti podatke o korisničkom imenu i lozinki. [7]

7.3. Potvrda autentičnosti

Model potvrde autentičnosti sličan je modelu osnovne HTTP autentifikacije koje je ranije objašnjen, osim što se u ovom modelu korisnički podaci šalju šifrirani. Klijent šalje zahtjev na zaštićen resurs i poslužitelj odgovara status kodom 401 i svojstvom „WWW-Authenticate“ u zaglavlju odgovora.

U nastavku je naveden primjer odgovora poslužitelja.

```
1. GET /protected_resource
2. 401 Unauthorized
3. WWW-Authenticate: Digest realm="Example Realm", nonce="P35kl89sdfghERT10Asdfnbvc",
   qop="auth"
```

U primjeru je vidljivo kako svojstvo „WWW-Authenticate“ specificira shemu provjere „Digest“ zajedno sa svojstvima „nonce“ i „qop“.

Nonce je proizvoljan niz znakova koji se koristi za kriptografske svrhe. Qop ili kvaliteta zaštite može sadržavati vrijednosti „auth“ ili „auth-int“. Auth označava da se provjera obavlja u svrhu provjere autentičnosti. Auth-int označava da se provjera koristi za provjeru autentičnosti i integriteta zahtjeva. [7]

Prilikom prijema odgovora od poslužitelja, ako je qop vrijednost postavljena na auth klijent šifrira korisniče podatke pomoću formule:

```
1. hash_value_1 = MD5(username:realm:password)
2. has_value_2 = MD5(request_method:request_uri)
3. digest = MD5(hash_value_1:nonce:hash_value_2)
```

Ukoliko je qop vrijednost postavljena na auth-int klijent šifrira korisniče podatke tako što uključuje i tijelo zahtjeva:

```
1. hash_value_1 = MD5(username:realm:password)
2. hash_value_2 = MD5(request_method:request_uri:MD5(request_body))
3. digest = MD5(hash_value_1:nonce:hash_value_2)
```

MD5 algoritam se koristi za izračunavanje hash vrijednosti. Šifrirani podaci su uključeni u zaglavlje i šalju se poslužitelju. Po primitku zahtjeva, poslužitelj dešifrira dobivene podatke i provjerava identitet korisnika. Nakon uspješne provjere poslužitelj dovršava zahtjev.

Ovaj model je sigurniji od osnovne HTTP autentifikacije jer se korisnički podaci nikad ne šalju u čistom tekstu. Međutim na SSL/TLS vezama i dalje je moguće da napadači pristupe šifriranim podacima. Isto tako budući da poslužitelj mora provjeriti podatke mora imati pristup nešifriranom korisničkom imenu i lozinki. Stoga poslužitelji mogu postati osjetljiviji na napade. [7]

7.4. Sigurnost temeljena na certifikatu

Ovaj model sigurnosti temelji se na certifikatima za potvrdu identiteta stranaka u komunikaciji. U komunikaciji koja se temelji na SSL/TLS vezi, klijent kao što je Web preglednik često provjerava identitet poslužitelja pomoću certifikata kako bi se osigurao da je poslužitelj ono što tvrdi da jest. Ovaj model se može proširiti za obavljanje međusobne provjere autentičnosti gdje poslužitelj može zatražiti certifikat od klijenta kao dio SSL/TLS postupka rukovanja i provjeriti identitet klijenta. [7]

U ovoj metodi, prilikom primanja zahtjeva za zaštićenim resursom, poslužitelj predstavlja svoj certifikat klijentu. Klijent provjerava da je certifikat poslužitelja ovlašten te šalje svoj certifikat poslužitelju. Poslužitelj provjerava korisnikov certifikat i ukoliko dođe do uspješne potvrde omogućiti će klijentu pristup do zaštićenog resursa. [7]

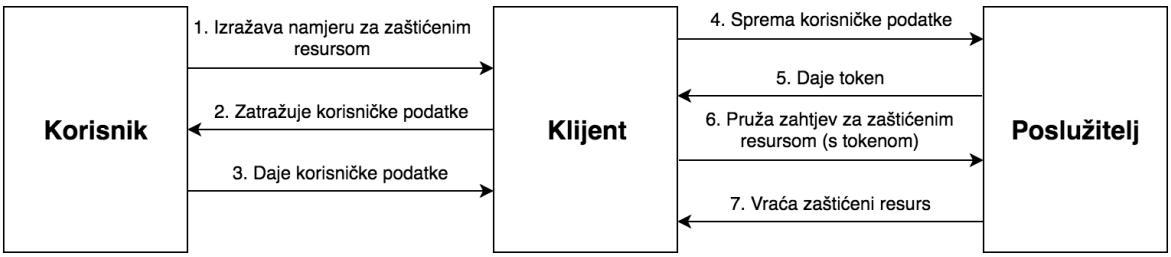
Model sigurnosti temeljen na certifikatu sigurniji je od modela slanja korisničkog imena i lozinke. Međutim implementacija i održavanje certifikata mogu biti skupi i obično se koriste u velikim sustavima.

7.5. XAuth

Kako su s vremenom REST Web servisi postali popularni, broj aplikacija trećih strana koje koriste te Web servise značajno raste. Te aplikacije trebaju korisničko ime i lozinku za interakciju s uslugama Web servisa i izvršavanje radnji u ime korisnika. To predstavlja veliki sigurnosni problem jer sada aplikacija treće strane imaju pristup korisničkim imenima i lozinkama. Rizik sigurnosti je taj što aplikacija treće strane može promijeniti korisnička imena i lozinke. Isto tako ako korisnik promijeni korisničko ime ili lozinku, treba ići na sve aplikacije treće strane i tamo ih također promijeniti. Ovaj mehanizam ne dopušta korisniku da prestane davati korisničko ime i lozinku aplikacijama treće strane. Jedina opcija u tom slučaju bi bila promjena lozinke.

XAuth sheme pružaju mehanizam za pristup zaštićenim resursima u ime korisnika bez potrebe za pohranjivanjem lozinki. U ovom pristupu aplikacija klijenta zatražila bi korisničko ime i lozinku od korisnika obično pomoću forme za prijavu. Klijent zatim šalje korisničko ime i lozinku poslužitelju. Poslužitelj prima korisničke podatke i provjerava ih. Nakon uspješne

provjere, token se vraća klijentu. Klijent pohranjuje token lokalno i dalje ga koristi kod pristupanja korisnikovom zaštićenom resursu. Klijent taj token uključuje u zahtjeve za resursom. To se obično postiže pomoću HTTP prilagođenog zaglavlja kao što je X-Auth-Token. Dugotrajnost tokena ovisi o njegovoj implementaciji. Token može vrijediti sve dok ga poslužitelj ne ukine ili valjanost tokena može isteći nakon nekog vremenskog perioda. [7]



Slika 13. XAuth sigurnosni tok (prema: Varanasi i Belida, 2015)

Aplikacije kao što je Twitter dopuštaju aplikacijama treće strane pristup svojem REST Web servisu koristeći XAuth shemu. Međutim, čak i uz XAuth aplikacija treće strane mora prvi puta dobiti korisničko ime i lozinku korisnika, što ostavlja mogućnost zloupotrebe. S obzirom na jednostavnost koja je uključena u XAuth, ovaj sigurnosni model je dobar kada jedna organizacija razvija REST Web servis i klijentsku aplikaciju. [7]

7.6. OAuth

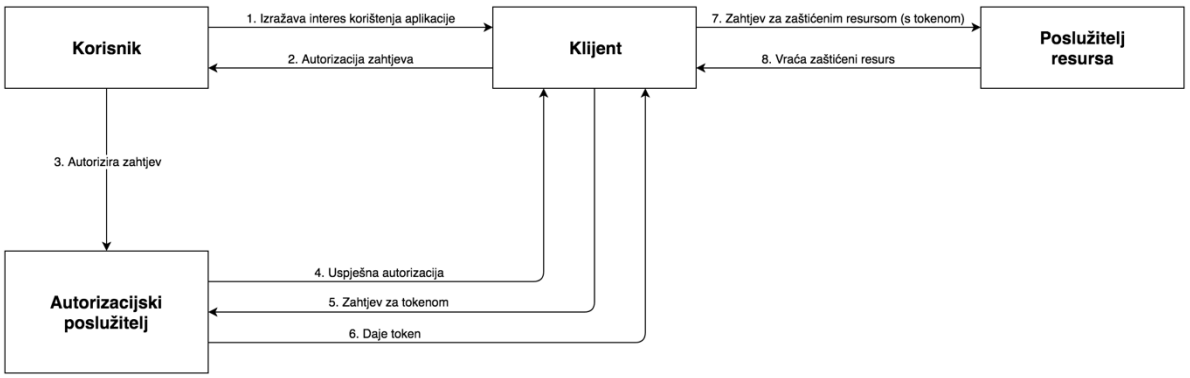
Otvorena autorizacija ili OAuth je programski okvir za pristup zaštićenim resursima u ime korisnika bez spremanja lozinke. OAuth je prvi puta predstavljen 2007. godine ali je zamijenjen OAuth verzijom 2.0 koja je predstavljena 2010. godine. [7]

OAuth 2.0 definira sljedeće četiri uloge:

- **Vlasnik resursa** – vlasnik resursa je korisnik koji želi dati pristup dijelovima svojeg računa ili resursa. Vlasnik resursa može biti Facebook ili Twitter korisnik
- **Klijent** – klijent je aplikacija koja želi pristupiti korisnikovim resursima. To može biti aplikacija treće strane koja želi pristupiti računu korisnika Twittera
- **Autorizacijski poslužitelj** – autorizacijski poslužitelj provjerava identitet korisnika i daje token klijentu za pristup resursima korisnika

- **Poslužitelj resursa** – poslužitelj resursa ima zaštićene korisničke resurse. Na primjer to bi bio Twitter Web servis za pristup twitovima ili vremenskoj crti korisnika i tako dalje

Interakcije između ovih uloga prikazane su na sljedećoj slici. OAuth 2.0 zahtjeva da se ta interakcija provodi putem SSL-a.



Slika 14. OAuth 2.0 sigurnosni tok (prema: Varanasi i Belida, 2015)

Prije nego što klijent može sudjelovati u toku koji je prikazan na prethodnoj slici mora se registrirati na autorizacijski poslužitelj. Za većinu javnih Web servisa poput Facebook ili Twitter to uključuje popunjavanje obrasca zahtjeva i pružanje informacija o klijentu, kao što su naziv aplikacije, domena aplikacije i web mjesto. Nakon uspješne registracije klijent će dobiti svoj ID klijenta i svoj tajni ključ. ID klijenta se koristi za jedinstveno identificiranje klijenta i javno je dostupan. [7]

OAuth interakcija započinje s korisnikom koji izražava zanimanje za upotrebom klijentske aplikacije, tj. aplikacije treće strane. Klijent zahtjeva autorizaciju za pristup zaštićenim resursima u ime korisnika i preusmjerava vlasnika resursa na autorizacijski poslužitelj. [7]

U nastavku je prikazan URI pomoću kojega klijent može vlasnika resursa preusmjeriti na autorizacijski poslužitelj.

```
1. https://example.com/authorize?client_id=CLIENT_ID&response_type=auth_code&call_back=
   CALL_BACK_URI&scope=read,tweet
```

Korištenje HTTPS-a je obavezno za svaku produkciju OAuth 2.0 interakcije. ID klijenta se koristi za identifikaciju klijenta na autorizacijskom poslužitelju. Parametar *scope* sadrži skup entiteta/uloga koje klijent treba.

Po primitku zahtjeva, autorizacijski poslužitelj bi dao izazov korisniku za provjeru autentičnosti, obično putem obrasca za prijavu. Korisnik tada daje svoje korisničko ime i lozinku. Nakon uspješne provjere korisničkih podataka autorizacijski poslužitelj preusmjerava u klijentsku aplikaciju koristeći parametar *call_back* iz zahtjeva. Autorizacijski poslužitelj također dodaje autorizacijski kod na vrijednost parametra *call_back*. [7]

U nastavku je prikazan URI kojega bi generirao autorizacijski poslužitelj.

```
1. https://example.com/code_callback?auth_code=6F99A74F2D066A267D6D838F88
```

Klijent zatim koristi autorizacijski kod da zatraži token od autorizacijskog poslužitelja. Kako bi to izveo, klijent u većini slučajeva poziva HTTP POST metodu na URI na način koji je prikazan u sljedećem primjeru.

```
1. https://example.com/access_token?client_id=CLIENT_ID&client_secret=CLIENT_SECRET&aut
   h_code=6F99A74F2D066A267D6D838F88
```

Kao što je vidljivo u primjeru, klijent šalje svoje podatke kao dio zahtjeva. Autorizacijski poslužitelj provjerava identitet klijenta i autorizacijski kod. Nakon uspješne provjere vraća token za pristup zaštićenom resursu. Primjer odgovora u JSON formatu je prikazan u nastavku.

```
1. {"access_token"="f292c6912e7710c8"}
```

Nakon primljenog tokena, klijent će zatražiti zaštićeni resurs od poslužitelja resursa koristeći token koji je dobio. Poslužitelj resursa provjerava token i nakon uspješne provjere daje zaštićeni resurs klijentu.

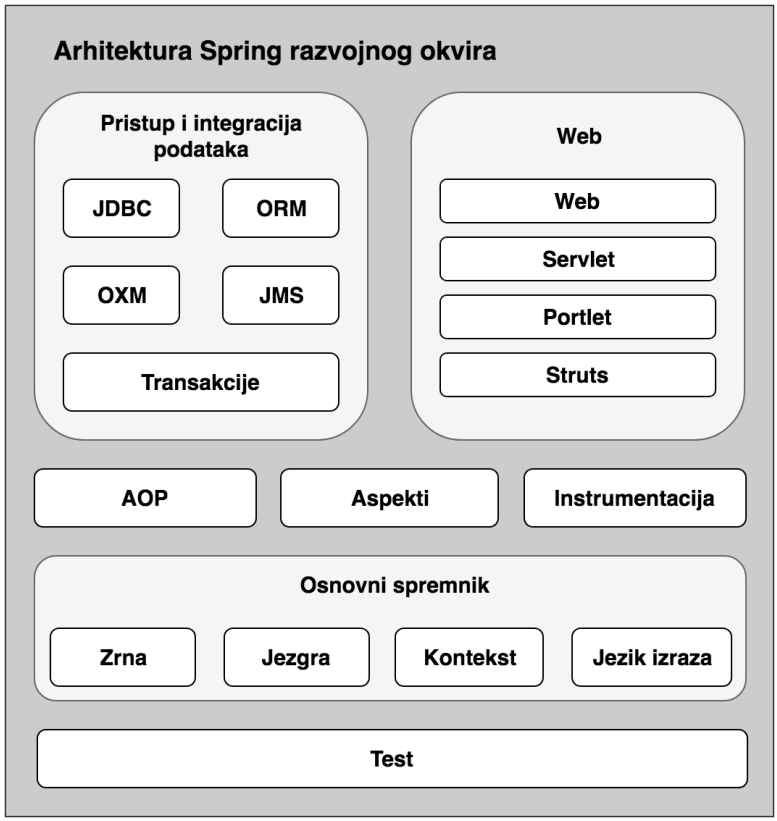
8. Praktični dio

U svrhu ovog diplomskog rada implementirane su dvije aplikacije. Prva aplikacija naziva *Get-Informations-Services* (u daljnjem tekstu poslužiteljska aplikacija) koja se nalazi na adresi http://get-information-services.herokuapp.com/ ima zadatak da pruža vlastite SOAP i REST Web servise te upravlja bazom podataka aplikacije. Druga aplikacija naziva *Get-Informations* (u daljnjem tekstu klijentska aplikacija) koja se nalazi na adresu http://get-information.herokuapp.com/ ima zadatak da koristi javne Web servise kao i Web servise koje pruža poslužiteljska aplikacija. Poslužiteljska kao i klijentska aplikacija implementirane su u Java programskom jeziku i Spring programskom okviru. Kao uzorak dizajna korišten je *Model View Controller* (u daljnjem tekstu MVC) jer je on jedan on najkorištenijih uzoraka dizajna u području Web programiranja u današnje vrijeme.

U nastavku će biti opisan Spring programski okvir, arhitektura sustava, Web servisi koji su korišteni u implementaciji klijentske aplikacije. Nadalje biti će prikazana rad klijentske aplikacije zajedno sa detaljnim opisom svake od funkcionalnosti aplikacije te će na kraju biti prikazani i objašnjeni na koji način su implementirane pojedine glavne stvari u poslužiteljskoj i klijentskoj aplikaciji.

8.1. Spring

Spring je programski okvir koji baziran na Java platformi koji pruža sveobuhvatnu infrastrukturu za razvoj Java aplikacija. Spring je modularan, te omogućuje da programeri koriste samo one dijelove koji su im potrebni. Spring se sastoji od značajki koje su ogranizirane u oko 20 modula. Na sljedećoj slici prikazani su njegovi glavni moduli te kako su oni grupirani:



Slika 15: Spring moduli (prema: https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/overview.html)

8.1.1. Pristup i integracija podataka

Sloj pristup i integracija podataka sastoji se od sljedećih modula:

- JDBC modul pruža apstrakcijski sloj koji uklanja potrebu za mukotrpnim JDBC programiranjem i raščlanjivanjem specifičnih kodova pogrešaka kod različitih proizvođača baze podataka.
- ORM modul pruža integracijske slojeve za popularne API-je koji su objektno mapirani, uključujući JPA, JDO, Hibernate i ostale. Koristeći ORM modul mogu se koristiti svi objektno orijentirani okviri u kombinaciji sa svim ostalim značajkama Springa.
- OXM modul pruža apstrakcijski sloj koji podržava implementaciju objektnog XML mapiranja za JAXB, Castor, XMLBeans, JiBX i XStream.
- JMS modul sadrži značajke za proizvodnju i konzumaciju poruka.
- Transakcijski modul podržava programirano i deklarativno upravljanje transakcijama za klase koje implementiraju posebna sučelja i za POJO klase. [12]

8.1.2. Web

Web sloj sastoji se od sljedećih modula:

- Spring Web modul pruža osnovne Web značajke integracije kao što su funkcionalnost višestrukog prijenosa datoteka i inicijalizaciju kontejnera IoC pomoću slušača servleta i web orijentiranog aplikacijskog konteksta.
- Spring Web Servlet modul sadrži primjenu Spring MVC uzorka za Web aplikacije. Spring MVC osigurava čistu odvojenost između modela, kontrolera i prikaza te se integrira sa svim ostalim značajkama Spring okvira.
- Spring Web Struts modul sadrži klase podrške za integraciju klasičnog Struts Web sloja unutar Spring aplikacije.
- Spring Web Portlet modul omogućuje implementaciju MVC uzorka u portlet okruženju i preslikava funkcionalnost Web Servlet modula [12]

8.1.3. Osnovni spremnik

Osnovni spremnik sastoji se od sljedećih modula:

- Moduli jezgra i zrna pružaju temeljne dijelove programskog okvira uključujući i *Dependency Injection*. Modul zrna sadrži *BeanFactory* što je zapravo sofisticirana implementacija uzorka *Factory Method*. Uklanja potrebu za programskim singletonima i omogućuje programerima razdvajanje konfiguracije i specifikaciju zavisnosti programske logike.
- Modul konteksta se temelji na čvrstoj bazi koju pružaju moduli jezgra i zrna. Modul konteksta nasljeđuje osobine iz modula zrna i daje podršku za internacionalizaciju, razmnožavanje događaja, učitavanje resursa i transparentno stvaranje konteksta kao što je servlet kontejner. Modul konteksta podržava i Java EE značajke kao što su EJB i JMX. *ApplicationContext* sučelje je žarišna točka modula konteskta.
- Modul jezik izraza pruža snažan jezik izraza za ispitivanje i manipuliranje grafikonom objekta pri izvođenju. To je nadopuna jedinstvenog jezika izražavanja kako je specificirano u JSP 2.1 specifikaciji. Jezik podržava postavljanje i pristupanje vrijednostima svojstva, dodjelu svojstva, pozivanje metoda, pristup kontekstu polja, kolekcija i indeksa. Također podržava i logičke i aritmetičke operacije, imenovane varijable i pronalaženje objekata prema nazivu iz Spring kontejnera za IoC. [12]

8.1.4. AOP i instrumentacija

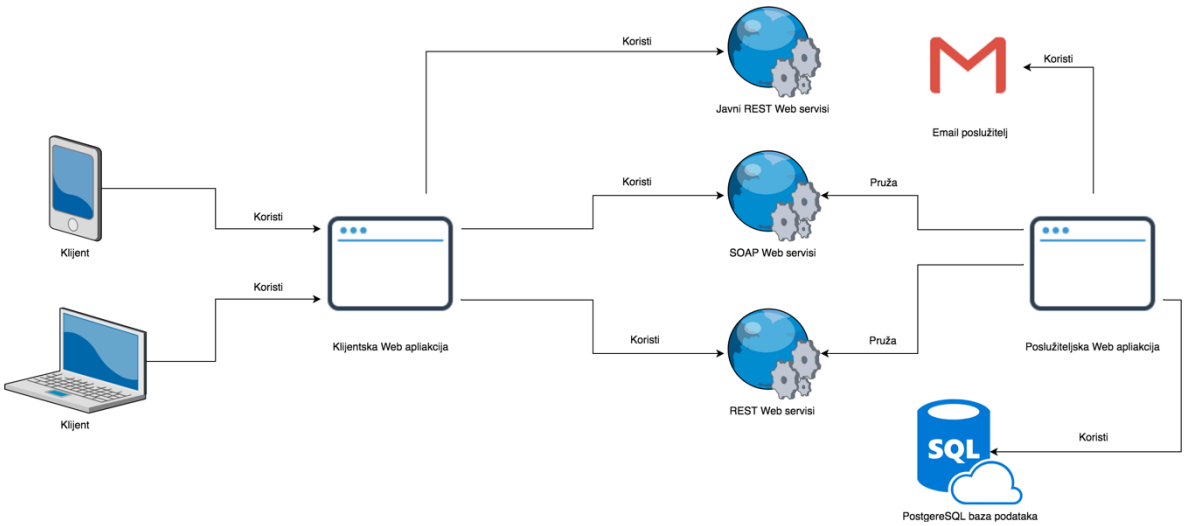
AOP modul pruža implementaciju programiranja usmjerenu na aspekte, što omogućuje definiranje presretača metoda za čisto odvajanje koda koji implementira funkcionalnost koja bi trebala biti odvojena.

Modul instrumentacije pruža klasičnu podršku instrumentacije klase i implementacije klastera koji se koriste u nekim aplikacijskim poslužiteljima. [12]

8.1.5. Test

Test modul podržava jedinično i integracijsko testiranje Spring komponenata pomoći JUnit ili TestNG programskih okvira. Također, omogućava učitavanje ApplicationContexta za korištenje u testovima i omogućuje kreiranje objekata koji oponašaju druge objekte

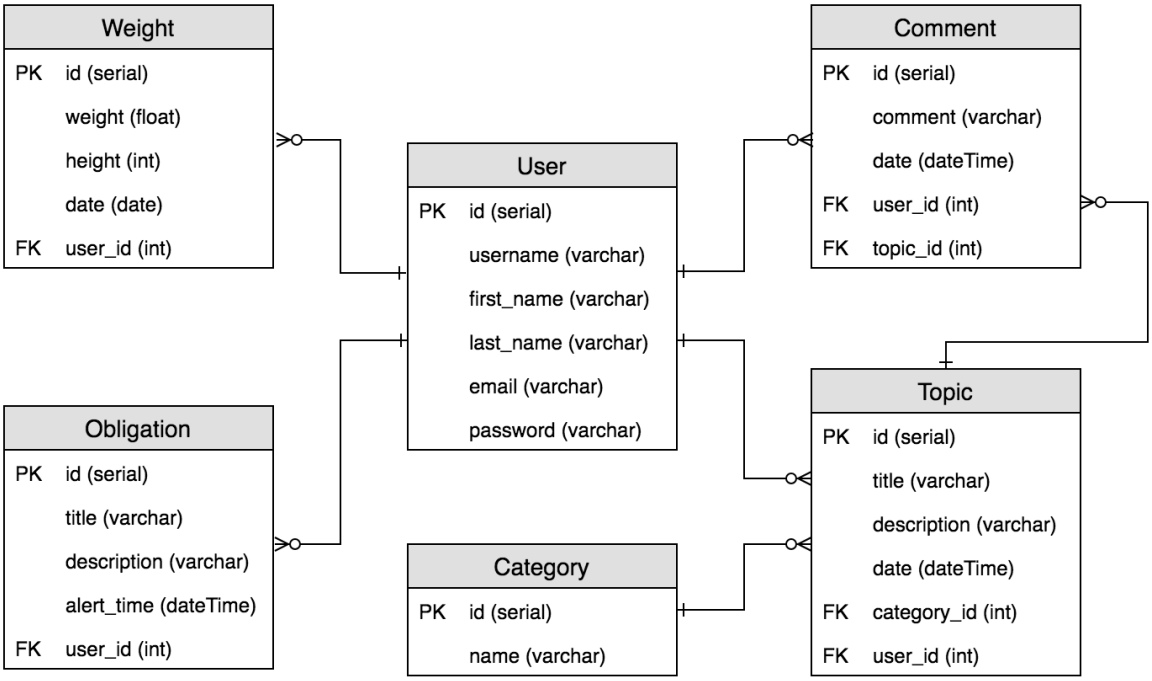
8.2. Arhitektura sustava



Slika 16: Arhitektura sustava (Izvor: Izrada autora)

Na prethodnoj slici je vidljivo kako korisnici mogu pristupiti klijentskoj aplikaciji za različitih uređaja koji imaju pristup Internetu, pošto se radi o Web aplikaciji. Klijentska aplikacija bazira se na komunikaciji s javnim Web servisima, te Web servisima koje pruža poslužiteljska aplikacija. Poslužiteljska aplikacija ima zadatak komuniciranja s PostreSQL bazom podataka,

slanje e-mail poruka preko e-mail poslužitelja te pružanje SOAP i REST Web servisa. Ti servisi će biti detaljnije opisani u sljedećem poglavlju.



Slika 17: ERA dijagram sustava (Izvor: Izrada autora)

Na prethodnoj slici prikazan je ERA dijagram PosgreSQL baze podataka koja je korištena u prethodno opisanom sustavu. Baza podataka sastoji se od šest tablica. Tablica *User* sadrži podatke o registriranim korisnicima sustava. Tablica *Obligation* sadrži podatke o obvezama korisnika. Svaka obaveza ima naziv, opis te vrijeme u koje se korisnika obavještava. Također tablica *Obligation* povezana je s tablicom *User* jer svaka obaveza pripada jednom korisniku, a jedan korisnik može imati više obaveza. Tablica *Weight* sadrži podatke o tjelesnoj masi i visini korisnika na određeni dan. Tablica *Weight* povezana je s tablicom *User* jer jedan zapis pripada jednom korisniku, dok jedan korisnik može imati više zapisa u tablici *Weight*. Tablica *Category* sadrži sve kategorije koje postoje u forumu. Tablica *Topic* sadrži podatke o objavi na forumu. Svaka objava se sastoji od naziva, opisa objave te datuma i vremena kada je objavljena. Također tablica *Topic* je povezana s tablicama *User* i *Category* iz razloga jer jedna objava pripada samo jednom korisniku i samo jednoj kategoriji dok jedan korisnik može objaviti više objava te jedna kategorija može sadržavati više objava. Tablica *Comment* sadrži podatke o komentaru na neku objavu, a on se sastoji od teksta komentara te datuma i vremena objave komentara. Tablica *Comment* povezana je s tablicama *User* i *Topic* iz razloga jer svaki

komentar ima korisnika koji ga je objavio, te svaki komentar pripada određenoj objavi. Također jedan korisnik može imati više komentara te jedna objava također može imati više komentara.

8.3. Opis Web servisa

Klijentska aplikacija za svoj rad koristi kolekciju javnih REST Web servisa kao i vlastito implementirane REST I SOAP Web servise koje pruža poslužiteljska aplikacija. U nastavku će oni biti detaljno objašnjeni.

8.3.1. Web servis za pretvorbu valuta

REST Web servis koji je korišten za pretvorbu valuta naziva je *Free Currency Converter API* te se njegova dokumentacija nalazi na https://www.currencyconverterapi.com/docs. U implementiranom sustavu korištene su njegove dvije metode, jedna za dohvaćanje svih svjetskih valuta te druga za pretvorbu iz jedne valute u drugu.

Metoda za dohvaćanje svih svjetskih valuta nalazi se na URI-u https://free.currencyconverterapi.com/api/v5/currencies. Za dobivanje liste svih svjetskih valuta potrebno je napraviti HTTP GET zahtjev na prethodno naveden URI. Rezultat Web servisa dobiven je u JSON formatu i ima sljedeću strukturu:

```
{
  "results": {
    "ALL": {
      "currencyName": "Albanian Lek",
      "currencySymbol": "Lek",
      "id": "ALL"
    },
    "XCD": {
      "currencyName": "East Caribbean Dollar",
      "currencySymbol": "$",
      "id": "XCD"
    },
    "EUR": {
      "currencyName": "Euro",
      "currencySymbol": "€",
      "id": "EUR"
    },
    ...
  }
}
```

Metoda koja omogućava pretvorbu valuta iz jedne u drugu nalazi se na URI-u https://free.currencyconverterapi.com/api/v5/convert. Za pretvorbu valuta potrebno je napraviti

HTTP GET zahtjev na prethodno naveden URI sa GET parametrom prikazanim u sljedećoj tablici.

Tablica 4: GET parametar zahtjeva REST Web servisa

Naziv parametra	Opis parametra	Primjer
q	Identifikator valute iz koje se pretvara i identifikator valute u koju se pretvara spojeni donjom crtom	EUR_HRK

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu i ima sljedeću strukturu:

```
1.  {
2.    "query": {
3.      "count": 1
4.    },
5.    "results": {
6.      "EUR_HRK": {
7.        "id": "EUR_HRK",
8.        "val": 7.442377,
9.        "to": "HRK",
10.       "fr": "EUR"
11.     }
12.   }
13. }
```

8.3.2. Web servis za pretvorbu adrese u geolokaciju

REST Web servis koji je korišten za pretvorbu adrese u geolokaciju je Google Geocoding API. Prije korištenja bilo kojeg Google Web servisa potrebno je registrirati aplikaciju tako da ona dobi svoj aplikacijski ključ. Nakon što se dobije aplikacijski ključ, on se koristi kod svakog zahtjeva na Google API. Detaljna dokumentacija Web servisa za pretvorbu adrese u geolokaciju nalazi se na https://developers.google.com/maps/documentation/geocoding/start. URI na koji se šalju zahtjevi je https://maps.googleapis.com/maps/api/geocode/json. Zahtjevi se šalju HTTP GET metodom na prije navedeni URI zajedno sa GET parametrima koji će biti opisani u sljedećoj tablici.

Tablica 5: GET parametri zahtjeva REST Web servisa

Naziv parametra	Opis parametra	Primjer
address	Adresa za koju se želi dobiti geolokaija	Pavlinska 2, Varaždin
key	Aplikacijski ključ	AIzaSyFMup7pevJQYQSB

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu te sadrži mnogo podataka od kojih se samo nekolicina koristi u implementiranom sustavu. Sljedeća tablica sadrži podatke koji su korišteni u implementiranom sustavu:

Tablica 6: Korišteni podaci odgovora REST Web servisa

Naziv podatka	Tip podatka	Opis podatka	Primjer
location	Object	Objekt koji sadrži geolokaciju tražene adrese	
lat	Double	Latituda tražene adrese	46.3076267
lng	Double	Longituda tražene adrese	16.3382566

(Izvor: Izrada autora)

8.3.3. Web servis za pretragu mjesta

REST Web servis koji omogućuje pretragu mjesta je *Google Place Search API* čija se dokumentacija nalazi na https://developers.google.com/places/web-service/search. Web servis omogućuje pretragu mjesta na puno načina ali u implementaciji sustava korištena je pretraga mjesta prema vrsti mjesta, geolokaciji te radijusu. URI na koji se šalje HTTP GET zahtjev je https://maps.googleapis.com/maps/api/place/nearbysearch/json. Također kod

slanja HTTP GET zahtjeva specificiraju se GET parametri koji su potrebni kako bi zahtjev bio valjan. Korišteni GET parametri opisani su u sljedećoj tablici.

Tablica 7: GET parametri zahtjeva REST Web servisa

Naziv parametra	Opis parametra	Primjer
location	Geolokacija za koju se vrši pretraga mjesta, u obliku *latitude,longitude*	46.3076267, 16.3382566
radius	Radius u kojem se vrši pretraga mjesta. Radius je specificiran u metrima od 0 do 50 000.	200
type	Tip mjesta prema kojemu se vrši pretraga	school
key	Aplikacijski ključ	AIzaSyFMup7pevJQYQSB

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu te sadrži mnogo podataka od kojih se samo nekolicina koristi u implementiranom sustavu. Sljedeća tablica sadrži podatke koji su korišteni u implementiranom sustavu:

Tablica 8: Korišteni podaci odgovora REST Web servisa

Naziv podatka	Tip podatka	Opis podatka	Primjer
results	Array	Polje objekata pronađenih mjesta	
name	String	Ime mjesta	FOI
place_id	String	Identifikator mjesta	NeqaEcRKOqp6ndM600

vicinity	String	Adresa mjesta	Pavlinska 2, Varaždin
lat	Double	Latituda mjesta	46.3076267
lng	Double	Longituda mjesta	16.3382566

(Izvor: Izrada autora)

8.3.4. Web servis za dobivanje detalja mjesta

REST Web servis za dobivanje detalja nekog mjesta je *Google Place Details API* čija se dokumentacija nalazi na https://developers.google.com/places/web-service/details. Jednom kada se dobi identifikator mjesta iz pretrage mjesta lako je dobiti detalje određenog mjesta prema njegovom identifikatoru. Potrebno je napraviti HTTP GET zahtjev na URI https://maps.googleapis.com/maps/api/place/details/json zajedno sa GET parametrima koji su navedeni u sljedećoj tablici.

Tablica 9: GET parametri zahtjeva REST Web servisa

Naziv parametra	Opis parametra	Primjer
placeid	Identifikator mjesta dobiven pretragom mjesta	ChIJdVmFOtiqaEcRFitgo-Jw1JU
key	Aplikacijski ključ	AIzaSyFMup7pevJQYQSB

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu te sadrži mnogo podataka od kojih se samo nekolicina koristi u implementiranom sustavu. Sljedeća tablica sadrži podatke koji su korišteni u implementiranom sustavu:

Tablica 10: Korišteni podaci odgovora REST Web servisa

Naziv podatka	Tip podatka	Opis podatka	Primjer
formatted_address	String	Adresa mjesta	Pavlinska 2, Varaždin
name	String	Ime mjesta	FOI
place_id	String	Jedinstveni identifikator mjesta	NeqaEcRKOqp6ndM600
international_phone_number	String	Internacionalni broj telefona	+385 42 390 804
lat	Double	Latituda mjesta	46.3076267
lng	Double	Longituda mjesta	16.3382566
raiting	Double	Rejting kojeg mjesto ima na Googlu	4.9
website	String	Web stranica mjesta	www.foi.unizg.hr
photos	Array	Polje objekata slika mjesta	
photo_reference	String	Jedinstveni identifikator slike mjesta	NeqaEcRKOqp6ndM600
weekday_text	Array	Polje koje sadrži sedam stringova koji označavaju radno vrijeme	Monday: 8 AM – 4 PM Tuesday: 8 AM – 4 PM Wednesday: 8 AM – 4 PM Thursday: 8 AM – 4 PM

		mjesta za svaki dan u tjednu	Friday: 8 AM – 4 PM Saturday: Closed Sunday: Closed

(Izvor: Izrada autora)

8.3.5. Web servis za dobivanje slika mjesta

Kako se pomoću REST Web servisa za dobivanje detalja mjesta ne dobivaju stvarne slike mjesta nego samo njihovi identifikatori na Googlu, potrebno je iskoristiti još jedan Google REST Web servis za dobivanje slika iz njihovih identifikatora. REST Web servis za dobivanje slika prema njihovim identifikatorima zove se *Google Places Photos API* i njegova dokumentacija se nalazi na https://developers.google.com/places/web-service/photos. Kako bi se dobila slika potrebno je napraviti HTTP GET zahtjev na URI https://maps.googleapis.com/maps/api/place/photo zajedno za GET parametrima koji su navedeni u sljedećoj tablici.

Tablica 11: GET parametri zahtjeva REST Web servisa

Naziv parametra	Opis parametra	Primjer
key	Aplikacijski ključ	AIzaSyFMup7pevJQYQSB
photoreference	Identifikator slike	NeqaEcRKOqp6ndM600
maxheight	Maksimalna visina slike	400

(Izvor: Izrada autora)

Rezultat ovog GET zahtjeva je slika sa zadanim identifikatorom maksimalne visine 400 piksela. Ukoliko je slika manja od 400 piksela ona će ostati u svojem izvornom oblika, a ukoliko je visina slike veća od 400 piksela slika će se skalirati na visinu od 400 piksela.

8.3.6. Web servis za dohvaćanje kulinarskih recepata

REST Web servis za dohvaćanje kulinarskih recepata je *EDAMAM Recipe Search API* te kako bi se došlo do njegove dokumentacije potrebno je prvo registrirati aplikaciju kako bi se dobili aplikacijski identifikator i aplikacijski ključ. Nakon što se dobe aplikacijski podaci za

pretragu kulinarskih recepata prema njihovim sastojcima potrebno je napraviti HTTP GET zahtjev na URI https://api.edamam.com/search zajedno sa GET parametrima koji su navedeni u sljedećoj tablici.

Tablica 12: GET parametri zahtjeva REST Web servisa

Naziv parametra	Opis parametra	Primjer
app_id	Identifikator aplikacije	AIzaSyFMup7pevJQYQSB
app_key	Aplikacijski ključ	NeqaEcRKOqp6ndM600
q	Popis sastojaka koji moraju biti uključeni u kulinarski recept	Chocolate

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu te sadrži mnogo podataka od kojih se samo nekolicina koristi u implementiranom sustavu. Sljedeća tablica sadrži podatke koji su korišteni u implementiranom sustavu:

Tablica 13: Korišteni podaci odgovora REST Web servisa

Naziv podatka	Tip podatka	Opis podatka	Primjer
hints	Array	Polje objekata recepta	
recipe	Object	Objekt recepta	
label	String	Naziv recepta	Iced chocolate
image	String	Link do slike recepta	https://www.edamam.com/web-img/7c1/7c7d7f.jpg
source	String	Izdavač recepta	David Lebovitz
shareAs	String	Link do recepta na EDAMAM Web stranici	www.edamam.com/recipe/iced-chocolate

yield	Integer	Broj serviranja	2
totalTime	Integer	Potrebno vrijeme za izradu (u min)	50
ingredientLines	Array	Polje koje sadrži stringove za svaki od sastojaka jela	3/4 cup (180ml) whole or lowfat milk 1 1/2 cups (6 ounces, 175g) ice cubes 8 chocolates or 4 ounces (115g) bittersweet or semisweet

(Izvor: Izrada autora)

8.3.7. Web servis za dohvaćanje vremenske progonoze

REST Web servis koji se koristi za dohvaćanje vremenske prognoze naziva je *OpenWeatherMap API*. REST Web servis pruža veliki izbor metoda sa različitim funkcionalnostima, ali u ovom radu korištene su njegove dvije metode, za dohvaćanje trenutne vremenske prognoze te za dohvaćanje vremenske prognoze za 5 dana. Kako bi bilo omogućeno korištene REST Web servisa potrebno se je obaviti registraciju kako bi se dobio aplikacijski ključ koji je potreban u daljnjim pozivanjima REST Web servisa.

Metoda za dohvaćanje trenutne vremenske prognoze pruža puno načina prema kojima je omogućeno dobivanje vremenske prognoze, npr. prema imenu grada, prema identifikatoru grada, prema geolokacijskim koordinatama, itd. U implementiranom sustavu korišteno je dohvaćanje trenutne vremenske prognoze prema geolokacijskim koordinatama. Dokumentacija REST Web servisa nalazi se na URI-u https://openweathermap.org/current. Kako bi se dobila trenutna vremenska prognoza potrebno je napraviti HTTP GET zahtjev na URI https://api.openweathermap.org/data/2.5/weather zajedno sa GET parametrima koji su navedeni u sljedećoj tablici.

Tablica 14: GET parametri zahtjeva REST Web servisa

Naziv parametra	Opis parametra	Primjer
appid	Aplikacijski ključ	AIzaSyFMup7pevJQYQSB

lat	Latituda mjesta	46.3076267
lon	Longituda mjesta	16.3382566
units	Jedinica mjere u kojima se prikazuje vremenska prognoza	metric

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu te sadrži mnogo podataka od kojih se samo nekolicina koristi u implementiranom sustavu. Sljedeća tablica sadrži podatke koji su korišteni u implementiranom sustavu:

Tablica 15: Korišteni podaci odgovora REST Web servisa

Naziv podatka	Tip podatka	Opis podatka	Primjer
name	String	Ime mjesta za koje se vremenska prognoza prikazuje	Varaždin
main	Array	Polje objekata koji sadrže glavne podatke o vremenu	
temp	Double	Trenutna vrijednost temperature	24.3
pressure	Double	Trenutna vrijednost tlaka zraka	1024
humidity	Integer	Trenutna vrijednost vlažnosti zraka	83

weather	Array	Polje objekata koji sadrže podatke o vremenu	
main	String	Naziv trenutnog vremena	Clouds
description	String	Opis trenutnog vremena	Broken clouds
icon	String	Identifikator ikone trenutnog vremena	04n
wind	Object	Objekt koji sadrži podatke o vjetru	
speed	Double	Brzina vjetra	5.1
rain	Object	Objekt koji sadrži podatke o kiši	
3h	Double	Količina kiše u zadnjih 3 sata	3.2

(Izvor: Izrada autora)

Metoda za dohvaćanje vremenske prognoze za 5 dana također pruža puno načina prema kojima je omogućeno dobivanje vremenske prognoze. Kao i u prethodnom slučaju kod implementacije sustava korišten je način dobivanja vremenske prognoze prema geolokacijskim koordinatama. Dokumentacija REST Web servisa nalazi se na URI-u https://openweathermap.org/forecast5. Kako bi se dobila vremenska prognoza za sljedećih 5 dana i to za svakih 3 sata potrebno je napraviti HTTP GET zahtjev na URI http://api.openweathermap.org/data/2.5/forecast sa GET parametrima koji su navedeni u sljedećoj tablici.

Tablica 16: GET parametri zahtjeva REST Web servisa

Naziv parametra	Opis parametra	Primjer
appid	Aplikacijski ključ	AIzaSyFMup7pevJQYQSB
lat	Latituda mjesta	46.3076267
lon	Longituda mjesta	16.3382566
units	Jedinica mjere u kojima se prikazuje vremenska prognoza	metric

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu te sadrži mnogo podataka od kojih se samo nekolicina koristi u implementiranom sustavu. Sljedeća tablica sadrži podatke koji su korišteni u implementiranom sustavu:

Tablica 17: Korišteni podaci odgvora REST Web servisa

Naziv podatka	Tip podatka	Opis podatka	Primjer
list	Array	Polje objekata koji sadrže vremenske podatke	
dt_txt	String	Datum i vrijeme za koji se vremenska prognoza odnosi	2018-09-03 09:00:00
name	String	Ime mjesta za koje se vremenska prognoza prikazuje	Varaždin

main	Array	Polje objekata koji sadrže glavne podatke o vremenu	
temp	Double	Vrijednost temperature	24.3
pressure	Double	Vrijednost tlaka zraka	1024
humidity	Integer	Vrijednost vlažnosti zraka	83
weather	Array	Polje objekata koji sadrže podatke o vremenu	
main	String	Naziv vremena	Clouds
description	String	Opis vremena	Broken clouds
icon	String	Identifikator ikone vremena	04n
wind	Object	Objekt koji sadrži podatke o vjetru	
speed	Double	Brzina vjetra	5.1
rain	Object	Objekt koji sadrži podatke o kiši	
3h	Double	Količina kiše u zadnjih 3 sata	3.2

(Izvor: Izrada autora)

8.3.8. Web servis za upravljanje korisnicima

REST Web servis za upravljanje korisnicima implementiran je u poslužiteljskoj aplikaciji i sadrži metode koje omogućuju kreiranje korisnika, dobivanje podataka za jednog korisnika te metodu za promjenu korisničkih podataka. Svim navedenim metodama moguće je pristupiti preko URI-a http://get-information-services.herokuapp.com/rest/user.

Korištenje metode za kreiranje korisnika moguće je pozivom HTTP POST metode na prethodno naveden URI zajedno sa tijelom zahtjeva koji mora biti u JSON formatu zapisa i mora sadržavati podatke koji su navedeni u sljedećoj tablici.

Tablica 18: Podaci koji se šalju kao tijelo POST zahtjeva na REST Web servis

Naziv podatka	Tip podatka	Opis podatka	Primjer
firstName	String	Ime korisnika	Pero
lastName	String	Prezime korisnika	Perić
email	String	Email	pero.perić@foi.hr
username	String	Korisničko ime	peroPerić
password	String	Lozinka	AIzaSyFMu

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu i ima sljedeći oblik.

```
1. {
2.     "status": "OK",
3.     "response": "",
4.     "error": ""
5. }
```

Korištenje metode za dobivanje podatka jednog korisnika moguće je pozivanjem HTTP GET metode na URI koji je naveden zajedno sa GET parametrom koji je naveden u sljedećoj tablici.

Tablica 19: GET parametar zahtjeva REST Web servisa

Naziv podatka	Tip podatka	Opis podatka	Primjer
username	String	Korisničko ime	peroPeric

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu i ima sljedeću strukturu:

```
{
    "status": "OK",
    "response": {
        "id": 4,
        "username": "peroPeric",
        "firstName": "Pero",
        "lastName": "Perić",
        "email": "pero_peric@foi.hr"
    },
    "error": ""
}
```

Korištenje metode za promjenu korisničkih podataka moguće je pristupiti pozivom HTTP PATCH metode na prethodno naveden URI zajedno sa tijelom zahtjeva koji mora biti u JSON formatu zapisa i mora sadržavati podatke koji su navedeni u sljedećoj tablici.

Tablica 20: Podaci koji se šalju kao tijelo PATCH zahtjeva REST Web servisa

Naziv podatka	Tip podatka	Opis podatka	Primjer
Id	Integer	Identifikacijski broj korisnika	4
firstName	String	Ime korisnika	Pero
lastName	String	Prezime korisnika	Perić
email	String	Email	pero.perić@foi.hr
username	String	Korisničko ime	peroPeric
password	String	Lozinka	AIzaSyFMu

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu i ima sljedeću strukturu:

```
{
    "status": "OK",
    "response": "",
    "error": ""
}
```

8.3.9. Web servis za upravljanje forumom

REST Web servis za upravljanje forumom implementiran je u poslužiteljskoj aplikaciji i sadrži 12 metoda koje predstavljaju CRUD operacije foruma. U nastavku će biti objašnjena svaka od njih.

Metodi za dobivanje svih kategorija foruma moguće je pristupiti pomoću HTTP GET zahtjeva na URI http://get-information-services.herokuapp.com/rest/forum/category. Rezultat Web servisa dobiven je u JSON formatu i ima sljedeću strukturu:

```
{
    "status": "OK",
    "response": [
        {
            "id": 1,
            "name": "Sport"
        },
        {
            "id": 2,
            "name": "IT"
        },
        {
            "id": 3,
            "name": "Politics"
        }
    ],
    "error": ""
}
```

Metodi za dodavanje nove kategorije foruma moguće je pristupiti pomoću HTTP POST zahtjeva na URI http://get-information-services.herokuapp.com/rest/forum/category zajedno sa tijelom zahtjeva koji je u kojemu se šalje ime kategorije u tekstualnom obliku. Rezultat Web servisa dobiven je u JSON obliku i ima sljedeću strukturu:

```
{
    "status": "OK",
    "response": "",
    "error": ""
}
```

Metodi za dobivanje svih objava jedne kategorije moguće je pristupiti pomoću HTTP GET zahtjeva na URI http://get-information-services.herokuapp.com/rest/forum/{category}/topic gdje je varijablu *{category}* potrebno zamijeniti imenom kategorije. Rezultat Web servisa je dobiven u JSON formatu i ima sljedeću strukturu:

```
{
    "status": "OK",
    "response": [
        {
            "id": 1,
            "title": "Premier league",
            "description": "Premier league start this week! I can't wait ;)",
            "date": 1534000283375,
            "category": {
                "id": 1,
                "name": "Sport"
            },
            "user": {
                "id": 2,
                "username": "admin",
                "firstName": "First",
                "lastName": "Last",
                "email": "admin@admin.com"
            }
        }
    ],
    "error": ""
}
```

Metodi za dodavanje nove objave pojedinoj kategoriji moguće je pristupiti pomoću HTTP POST zahtjeva na URI http://get-information-services.herokuapp.com/rest/forum/topic zajedno sa tijelom zahtjeva koje mora biti u JSON formatu i sadržavati podatke navedene u sljedećoj tablici:

Tablica 21: Podaci koji se šalju kao tijelo POST zahtjeva REST Web servisa

Naziv podatka	Tip podatka	Opis podatka	Primjer
username	String	Korisničko ime korisnika koji kreira objavu	peroPerić
category	String	Ime kategorije	Sport
title	String	Naslov objave	Bundesliga

description	String	Opis objave	Bundesliga will be super interesting this year.

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu i ima sljedeću strukturu:

```
{
    "status": "OK",
    "response": "",
    "error": ""
}
```

Metodi za dobivanje svih podataka jedne objave moguće je pristupiti pomoću HTTP GET zahtjeva na URI http://get-information-services.herokuapp.com/rest/forum/topic zajedno sa GET parametrom koji je naveden u sljedećoj tablici.

Tablica 22: GET parametar zahtjeva REST Web servisa

Naziv podatka	Opis podatka	Primjer
Id	Identifikacijski broj objave	1

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu i ima sljedeću strukturu:

```
{
    "status": "OK",
    "response": {
        "id": 1,
        "title": "Premier league",
        "description": "Premier league start this week! I can't wait ;)",
        "date": 1534000283375,
        "category": {
            "id": 1,
            "name": "Sport"
        },
        "user": {
            "id": 2,
            "username": "admin",
            "firstName": "First",
            "lastName": "Last",
            "email": "admin@admin.com"
        }
    },
```

```
    "error": ""
}
```

Metodi za promjenu podataka određene objave moguće je pristupiti pomoću HTTP PATCH zahtjeva na URI http://get-information-services.herokuapp.com/rest/forum/topic te mora sadržavati tijelo zahtjeva u JSON formatu i imati podatke koji su navedeni u sljedećoj tablici.

Tablica 23: Podaci koji se šalju kao tijelo PATCH zahtjeva REST Web servisa

Naziv podatka	Tip podatka	Opis podatka	Primjer
id	Integer	Identifikacijski broj objave	1
title	String	Naslov objave	Bundesliga
description	String	Opis objave	Bundesliga will be super interesting this year.

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu i ima sljedeću strukturu:

```
{
    "status": "OK",
    "response": "",
    "error": ""
}
```

Metodi za brisanje podataka jedne objave moguće je pristupiti pomoću HTTP DELETE zahtjeva na URI http://get-information-services.herokuapp.com/rest/forum/topic zajedno sa GET parametrom koji je naveden u sljedećoj tablici.

Tablica 24: GET parametar zahtjeva REST Web servisa

Naziv podatka	Tip podatka	Opis podatka	Primjer
Id	Integer	Identifikacijski broj objave	1

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu i ima sljedeću strukturu:

```
{
    "status": "OK",
    "response": "",
    "error": ""
}
```

Metodi za dobivanje svih komentara jedne objave moguće je pristupiti pomoću HTTP GET zahtjeva na URI http://get-information-services.herokuapp.com/rest/forum/{topic}/comments gdje je varijablu *{topic}* potrebno zamijeniti identifikacijskim brojem objave. Rezultat Web servisa dobiven je u JSON formatu i ima sljedeću strukturu:

```
{
    "status": "OK",
    "response": [
        {
            "id": 1,
            "comment": "No way. Borussia Dortmund have beter team this year.",
            "date": 1535881101155,
            "user": {
                "id": 1,
                "username": "user",
                "firstName": "First",
                "lastName": "Last",
                "email": "user@user.com"
            },
            "topic": {
                "id": 2,
                "title": "Bundesliga",
                "description": "Bundesliga will be super interesting this year. Baye
rn Munchen must be a champion.",
                "date": 1535880955711,
                "category": {
                    "id": 1,
                    "name": "Sport"
                },
                "user": {
                    "id": 3,
                    "username": "ivic",
                    "firstName": "Ivo",
                    "lastName": "Ivić",
                    "email": "ivic.ivic@foi.hr"
                }
            }
        }
    ],
    "error": ""
}
```

Metodi za dobivanje svih podataka jednog komentara moguće je pristupiti pomoću HTTP GET zahtjeva na URI http://get-information-services.herokuapp.com/rest/forum/comment zajedno sa GET parametrom koji je naveden u sljedećoj tablici.

Tablica 25: GET parametar zahtjeva REST Web servisa

Naziv podatka	Tip podatka	Opis podatka	Primjer
Id	Integer	Identifikacijski broj objave	1

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu i ima sljedeću strukturu:

```
1.  {
2.      "status": "OK",
3.      "response": {
4.          "id": 1,
5.          "comment": "No way. Borussia Dortmund have beter team this year.",
6.          "date": 1535881101155,
7.          "user": {
8.              "id": 1,
9.              "username": "user",
10.             "firstName": "First",
11.             "lastName": "Last",
12.             "email": "user@user.com"
13.         },
14.         "topic": {
15.             "id": 2,
16.             "title": "Bundesliga",
17.             "description": "Bundesliga will be super interesting this year. Bayern M
    unchen must be a champion.",
18.             "date": 1535880955711,
19.             "category": {
20.                 "id": 1,
21.                 "name": "Sport"
22.             },
23.             "user": {
24.                 "id": 3,
25.                 "username": "ivic",
26.                 "firstName": "Ivo",
27.                 "lastName": "Ivić",
28.                 "email": "ivic.ivic@foi.hr"
29.             }
30.         }
31.     },
32.     "error": ""
33. }
```

Metodi za dodavanje novog komentara moguće je pristupit pomoću HTTP POST zahtjeva na URI http://get-information-services.herokuapp.com/rest/forum/comment zajedno sa tijelom zahtjeva koje mora biti u JSON formatu i imati podatke koji su navedeni u sljedećoj tablici.

Tablica 26: Podaci koji se šalju ako tijelo POST zahtjeva REST Web servisa

Naziv podatka	Tip podatka	Opis podatka	Primjer
username	String	Korisničko ime korisnika koji dodaje komentar	peroPerić
topicId	Integer	Identifikacijski broj objave	2
comment	String	Tekst komentara	No way. Borussia Dortmund have beter team this year.

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu i ima sljedeću strukturu:

```
1. {
2.     "status": "OK",
3.     "response": "",
4.     "error": ""
5. }
```

Metodi za promjenu sadržaja komentara moguće je pristupiti pomoću HTTP PATCH zahtjeva na URI http://get-information-services.herokuapp.com/rest/forum/comment zajedno sa tijelom zahtjeva koji mora biti u JSON formatu i imati podatke koji su navedeni u sljedećoj tablici:

Tablica 27: Podaci koji se šalju kao tijelo PATCH zahtjeva REST Web servisa

Naziv podatka	Tip podatka	Opis podatka	Primjer
id	Integer	Identifikacijski broj komentara	1
comment	String	Tekst komentara	Borussia Dortmund have beter team this year.

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu i ima sljedeću strukturu:

```
{
    "status": "OK",
    "response": "",
    "error": ""
}
```

Metodi za brisanje podataka jednog komentara moguće je pristupiti pomoću HTTP DELETE zahtjeva na URI http://get-information-services.herokuapp.com/rest/forum/comment zajedno za GET parametrom koji je naveden u sljedećoj tablici:

Tablica 28: GET parametar zahtjeva REST Web servisa

Naziv podatka	Tip podatka	Opis podatka	Primjer
Id	Integer	Identifikacijski broj komentara	1

(Izvor: Izrada autora)

Rezultat Web servisa dobiven je u JSON formatu i ima sljedeću strukturu:

```
{
    "status": "OK",
    "response": "",
    "error": ""
}
```

8.3.10. Web servis za upravljanje obavezama

SOAP Web servis za upravljanje korisničkim obavezama implementiran je u poslužiteljskoj aplikaciji i pruža dvije metode, metodu za kreiranje nove obaveze te metodu za dohvaćanje svih nadolazećih obaveza korisnika.

Metodi za kreiranje nove obaveze moguće je pristupiti pomoću krajnje točke koja se nalazi na URI-u http://get-information-services.herokuapp.com/soap. SOAP poruka zahtjeva mora biti pisana u XML formatu i imati sljedeći oblik:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:art="http://www.GetInformationsServices.foi.unizg.com/soap">
   <soapenv:Header/>
   <soapenv:Body>
      <art:addObligationRequest>
```

```
            <art:username>peroPeric</art:username>
            <art:title>Obrana diplomskog rada</art:title>
            <art:descriptions>
               Obrana diplomskog rada u kabinetu mentora.
            </art:descriptions>
            <art:alertTime>2018-09-25T09:00:00</art:alertTime>
         </art:addObligationRequest>
      </soapenv:Body>
   </soapenv:Envelope>
```

SOAP poruka odgovora dobivena je u XML formatu i ima sljedeću strukturu:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:addObligationResponse
            xmlns:ns2="http://www.GetInformationsServices.foi.unizg.com/soap">
            <ns2:statusCode>OK</ns2:statusCode>
        </ns2:addObligationResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Metodi za dobivanje svih nadolazećih obaveza korisnika moguće je pristupiti pomoću krajnje točke koja se nalazi na URI-u http://get-information-services.herokuapp.com/soap. SOAP poruka zahtjeva mora biti pisana u XML formatu i imati sljedeći oblik:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:art="http://www.GetInformationsServices.foi.unizg.com/soap">
   <soapenv:Header/>
   <soapenv:Body>
      <art:getObligationsByUserRequest>
         <art:username>peroPeric</art:username>
      </art:getObligationsByUserRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP poruka odgovora dobivena je u XML formatu i ima sljedeću strukturu:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:getObligationsByUserResponse
            xmlns:ns2="http://www.GetInformationsServices.foi.unizg.com/soap">
            <ns2:statusCode>OK</ns2:statusCode>
            <ns2:obligation>
                <ns2:title>Obrana diplomskog rada</ns2:title>
                <ns2:description>
                   Obrana diplomskog rada u kabinetu mentora.
                </ns2:description>
                <ns2:alertTime>2018-09-25T09:00:00</ns2:alertTime>
            </ns2:obligation>
        </ns2:getObligationsByUserResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

8.3.11. Web servis za upravljanje težinom

SOAP Web servis za upravljanje težinom implementiran je u poslužiteljskoj aplikaciji i pruža tri metode, metodu za dodavanje nove težine, metodu za dohvaćanje zadnjih podataka o težini te metodu za dohvaćanje podataka o težini u zadanom vremenskom razdoblju.

Metodi za dodavanje nove težine moguće je pristupiti pomoću krajnje točke koja se nalazi na URI-u http://get-information-services.herokuapp.com/soap. SOAP poruka zahtjeva mora biti pisana u XML formatu i imati sljedeći oblik:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:art="http://www.GetInformationsServices.foi.unizg.com/soap">
   <soapenv:Header/>
   <soapenv:Body>
      <art:addWeightRequest>
         <art:username>peroPeric</art:username>
         <art:weight>78.5</art:weight>
         <art:height>179</art:height>
      </art:addWeightRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP poruka odgovora dobivena je u XML formatu i ima sljedeću strukturu:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:addWeightResponse
            xmlns:ns2="http://www.GetInformationsServices.foi.unizg.com/soap">
            <ns2:statusCode>OK</ns2:statusCode>
        </ns2:addWeightResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Metodi za dobivanje zadnjih podataka o težini za zadanog korisnika moguće je pristupiti pomoću krajnje točke na URI-u http://get-information-services.herokuapp.com/soap. SOAP poruka zahtjeva mora biti pisana u XML formatu i imati sljedeći oblik:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:art="http://www.GetInformationsServices.foi.unizg.com/soap">
   <soapenv:Header/>
   <soapenv:Body>
      <art:getLastWeightByUserRequest>
         <art:username>peroPeric</art:username>
      </art:getLastWeightByUserRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP poruka odgovora dobivena je u XML formatu i ima sljedeću strukturu:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:getLastWeightByUserResponse
            xmlns:ns2="http://www.GetInformationsServices.foi.unizg.com/soap">
            <ns2:statusCode>OK</ns2:statusCode>
            <ns2:weight>
                <ns2:date>2018-09-08Z</ns2:date>
                <ns2:weight>78.5</ns2:weight>
                <ns2:height>179</ns2:height>
                <ns2:bmi>24.49986</ns2:bmi>
            </ns2:weight>
        </ns2:getLastWeightByUserResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Metodi za dobivanje podataka o težini korisnika u zadanom vremenskom razdoblju moguće je pristupiti preko krajnje točne na URI-u http://get-information-services.herokuapp.com/soap. SOAP poruka zahtjeva mora biti pisana u XML formatu i imati sljedeći oblik:

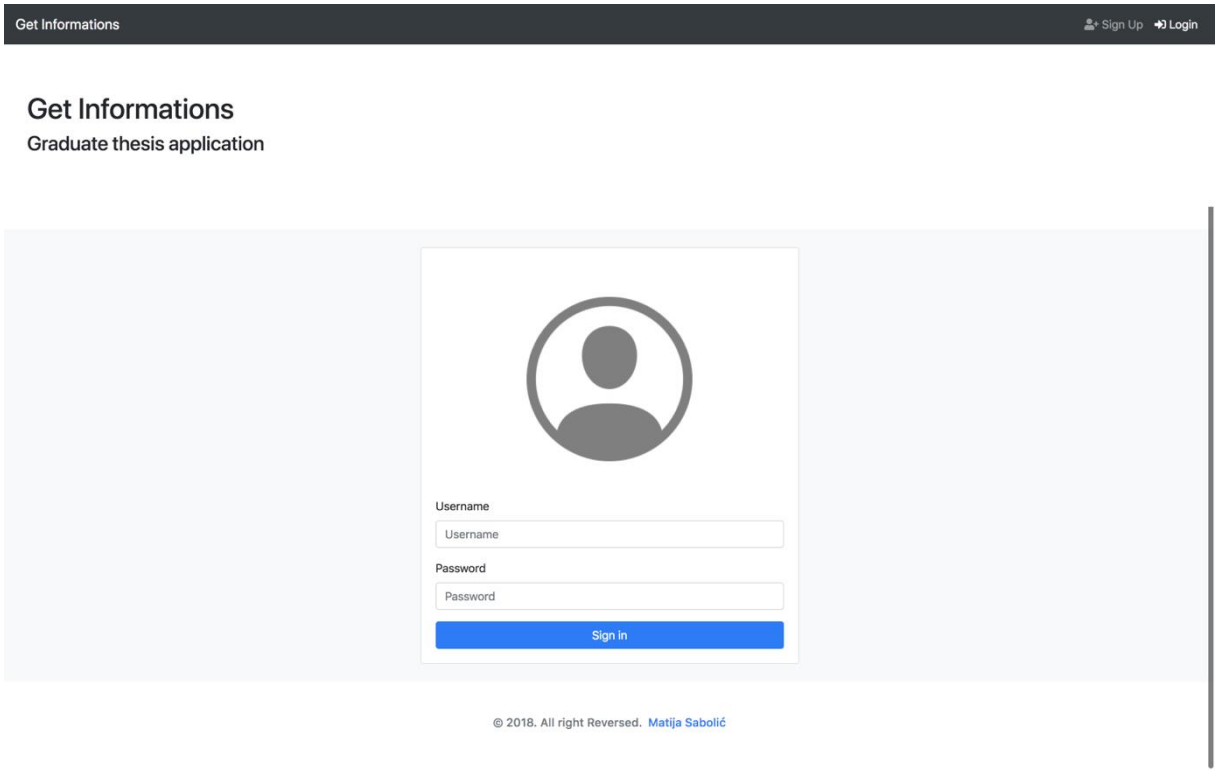
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:art="http://www.GetInformationsServices.foi.unizg.com/soap">
   <soapenv:Header/>
   <soapenv:Body>
      <art:getWeightListRequest>
         <art:username>ivic</art:username>
         <art:from>2018-09-01Z</art:from>
         <art:to>2018-09-10Z</art:to>
      </art:getWeightListRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP poruka odgovora dobivena je u XML formatu i ima sljedeću strukturu:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:getWeightListResponse
            xmlns:ns2="http://www.GetInformationsServices.foi.unizg.com/soap">
            <ns2:statusCode>OK</ns2:statusCode>
            <ns2:weight>
                <ns2:date>2018-09-02Z</ns2:date>
                <ns2:weight>95.0</ns2:weight>
                <ns2:height>179</ns2:height>
                <ns2:bmi>29.649513</ns2:bmi>
            </ns2:weight>
            <ns2:weight>
                <ns2:date>2018-09-02Z</ns2:date>
                <ns2:weight>92.0</ns2:weight>
                <ns2:height>179</ns2:height>
                <ns2:bmi>28.713213</ns2:bmi>
            </ns2:weight>
            <ns2:weight>
```

```
                <ns2:date>2018-09-02Z</ns2:date>
                <ns2:weight>90.0</ns2:weight>
                <ns2:height>179</ns2:height>
                <ns2:bmi>28.089012</ns2:bmi>
            </ns2:weight>
            <ns2:weight>
                <ns2:date>2018-09-02Z</ns2:date>
                <ns2:weight>87.0</ns2:weight>
                <ns2:height>179</ns2:height>
                <ns2:bmi>27.152712</ns2:bmi>
            </ns2:weight>
            <ns2:weight>
                <ns2:date>2018-09-08Z</ns2:date>
                <ns2:weight>81.5</ns2:weight>
                <ns2:height>179</ns2:height>
                <ns2:bmi>25.436161</ns2:bmi>
            </ns2:weight>
        </ns2:getWeightListResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

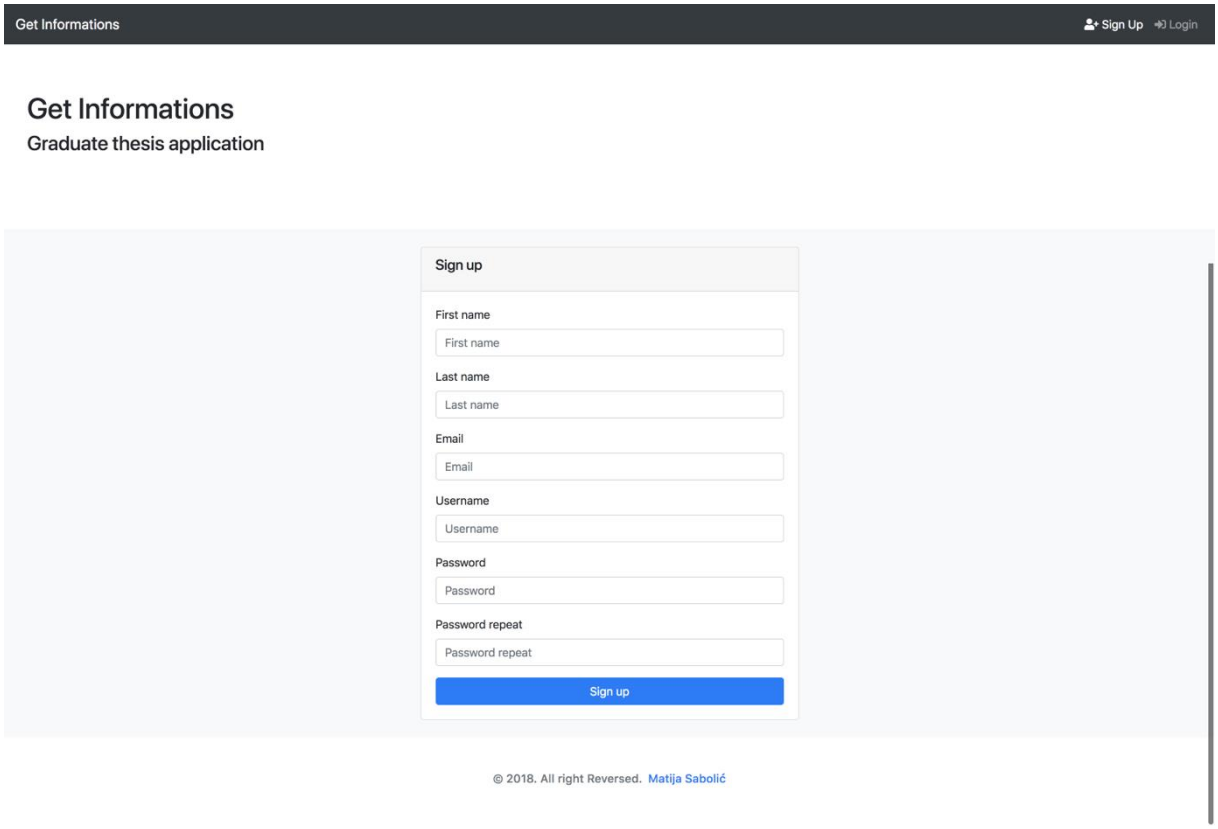
8.4. Prikaz rada sustava



Slika 18: Stranica za prijavu u aplikaciju (Izvor: Izrada autora)

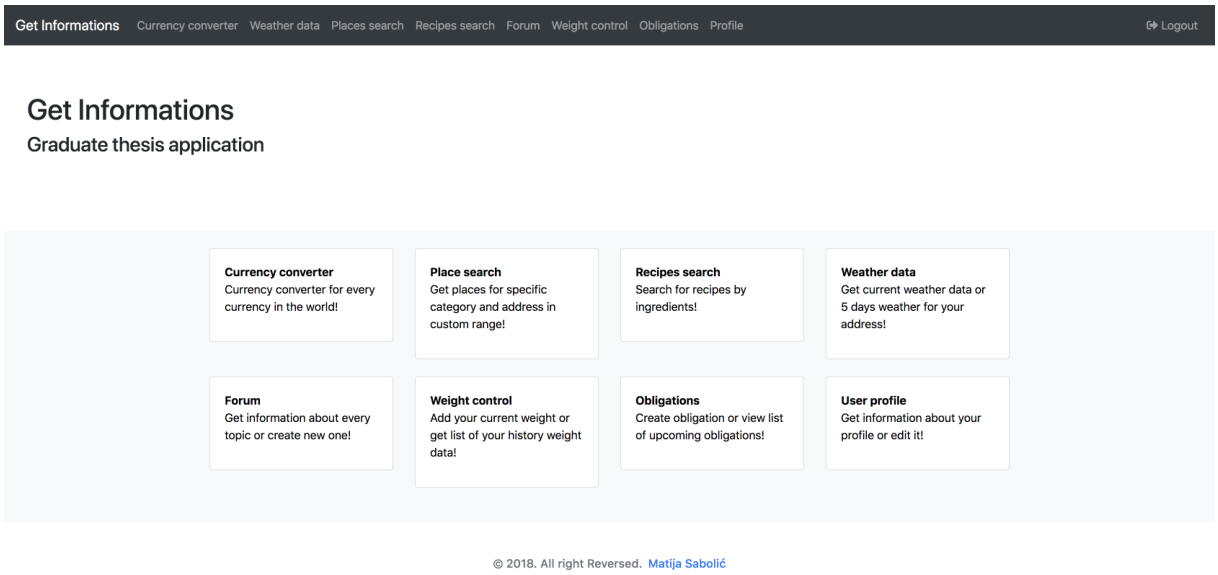
Svaki korisnik aplikacije za rad u njoj mora se prijaviti preko stranice za prijavu pomoću korisničkog imena i lozinke. Uneseni podaci se zatim provjeravaju sa podacima u bazi podataka te ukoliko je korisnik unio ispravne podatke nastavlja s radom u aplikaciji. Ukoliko

korisnik nema svoje korisničke podatke mora napraviti registraciju u sustav putem forme za registraciju. Registracijska forma prikazana je na sljedećoj slici.



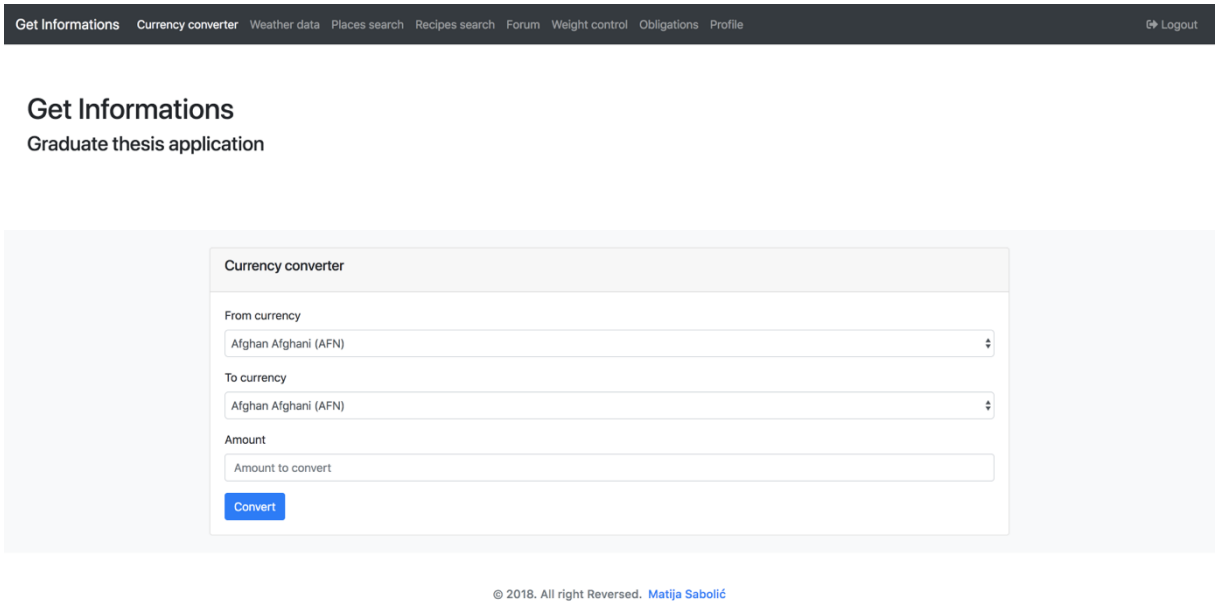
Slika 19: Stranica za registraciju korisnika (Izvor: Izrada autora)

Nakon što se popuni registracijska forma upisani podaci se šalju REST Web servisu poslužiteljske aplikacije. REST Web servis provjerava ukoliko u bazi podataka ne postoji korisnik s unesenim korisničkim imenom ili email adresom. Ukoliko ne postoji niti jedan korisnik u bazi podataka s tim korisničkim imenom ili email adresom uneseni podaci se spremaju u bazu podataka i korisnik se uspješno registrirao te REST Web servis vraća poruku o uspješnom registriranju. Ukoliko se uneseni podaci podudaraju s nekima već zapisanima u bazi podataka REST Web servis vraća opis greške te se isti prikazuje korisniku na registracijskoj formi. Nakon što se korisnik uspješno registrirao u sustav može se s svojim korisničkim podacima prijaviti u isti. Nakon uspješne prijave korisnik se preusmjerava na početnu stranicu aplikacije koja je prikazana na sljedećoj slici.



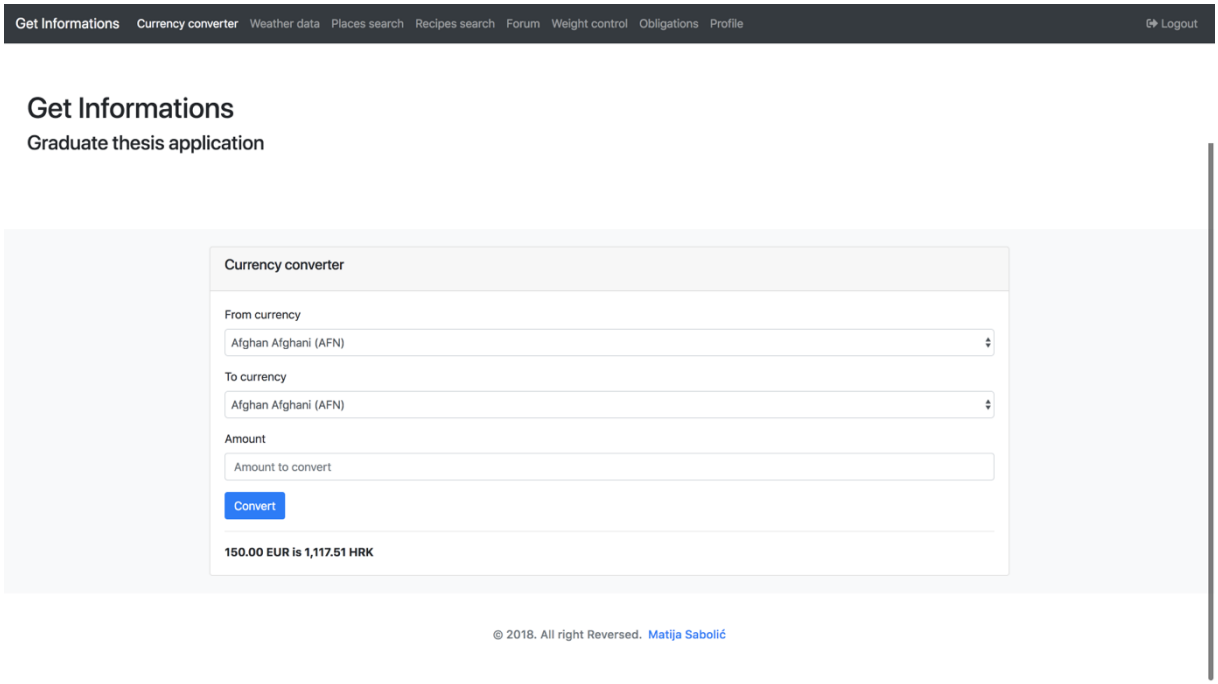
Slika 20: Početna stranica korisničke aplikacije (Izvor: Izrada autora)

Početna stranica aplikacije sadrži poveznice na sve dijelove aplikacije koji se također mogu pronaći i u navigacijskoj traci. U nastavku će biti objašnjena pojedina funkcionalnost aplikacije.



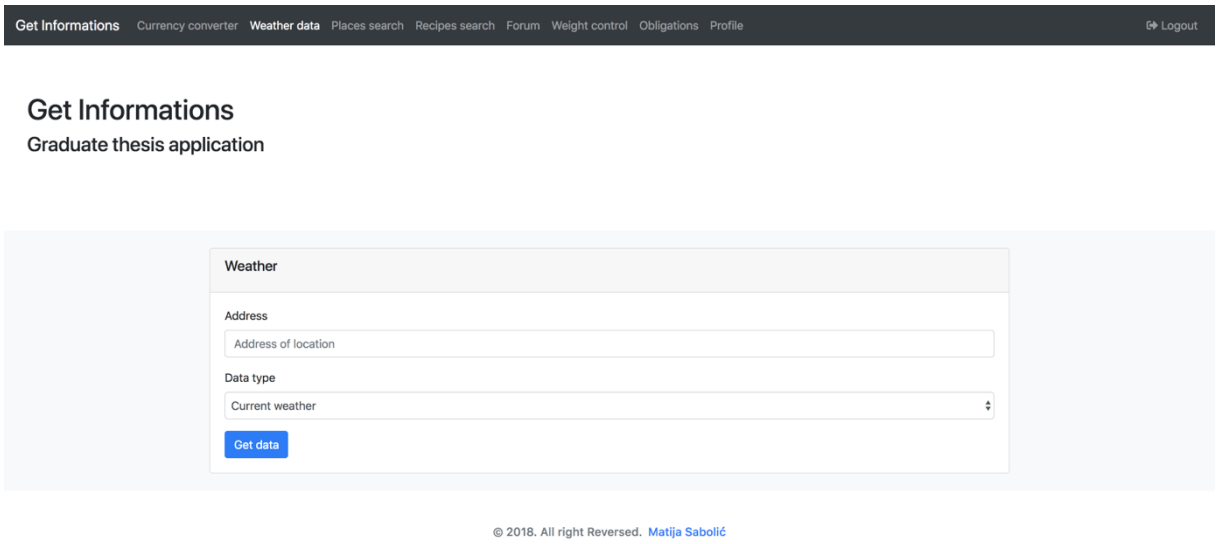
Slika 21: Stranica za pretvorbu valuta (Izvor: Izrada autora)

Prva funkcionalnost aplikacije je pretvorba valuta. Pretvorba se obavlja na način da korisnik iz padajućeg izbornika odabere iz koje valute želi pretvorbu u koju valutu te upiše željeni iznos za pretvorbu. Padajući izbornici sadrže podatke koji se dobe putem REST Web servisa za dobivanje svih svjetskih valuta. Kada korisnik popuni sve potrebne podatke klikom na gumb *Convert* u pozadini se poziva REST Web servis za pretvorbu valuta. Nakon što REST Web servis vrati rezultat on se prikazuje korisniku na zaslonu. Prikaz rezultata prikazan je na sljedećoj slici.



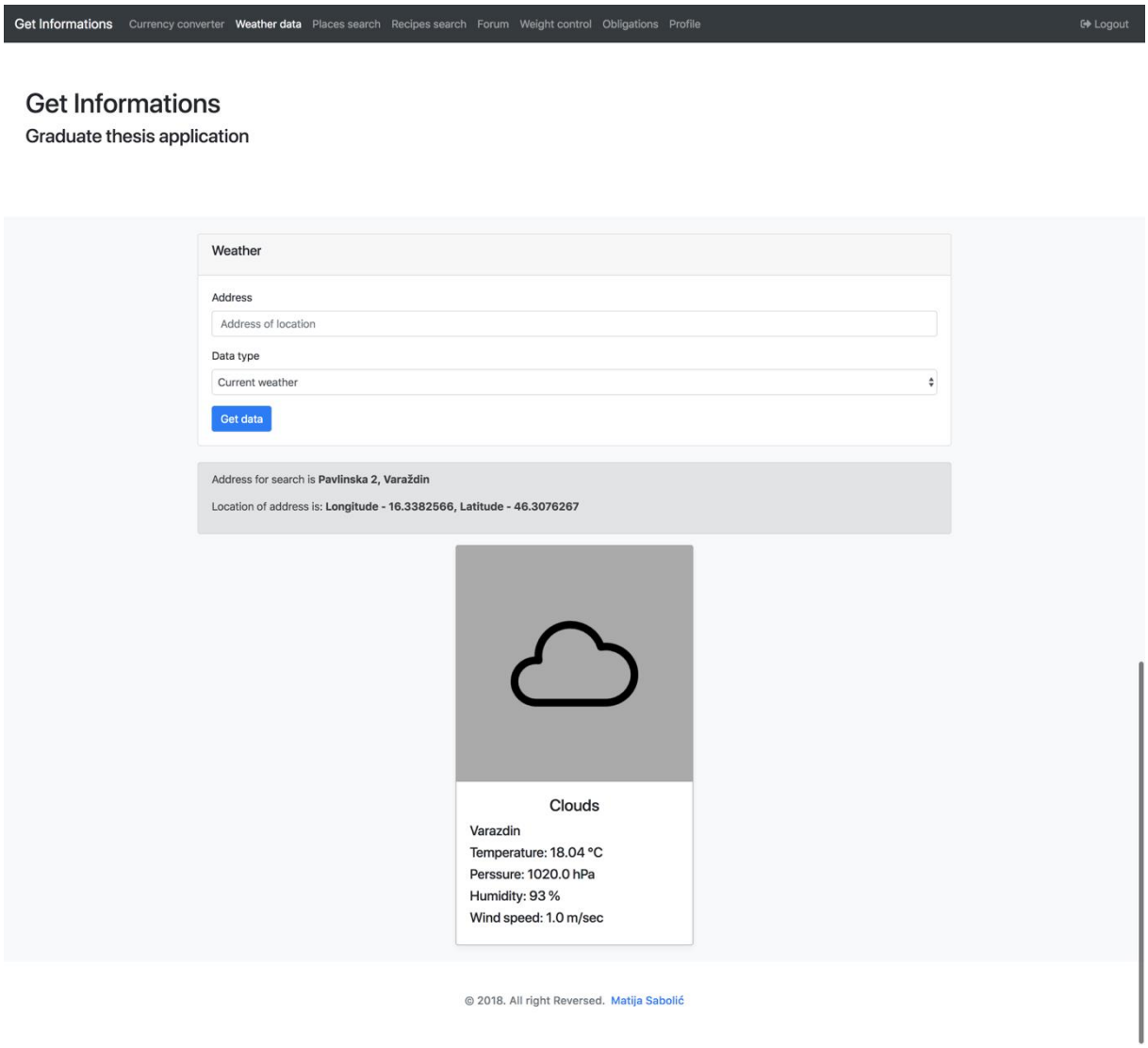
Slika 22: Prikaz rezultata pretvorbe valuta (Izvor: Izrada autora)

Druga funkcionalnost aplikacije je dobivanje vremenske prognoze za određenu adresu. Korisnik upisuje adresu za koju želi dobiti vremensku prognozu te odabire iz padajućeg izbornika želi li dobiti trenutnu vremensku prognozu ili vremensku prognozu za sljedećih 5 dana. Sljedeća slika prikazuje stranicu za unos tih podataka.



Slika 23: Stranica za dobivanje vremenske prognoze (Izvor: Izrada autora)

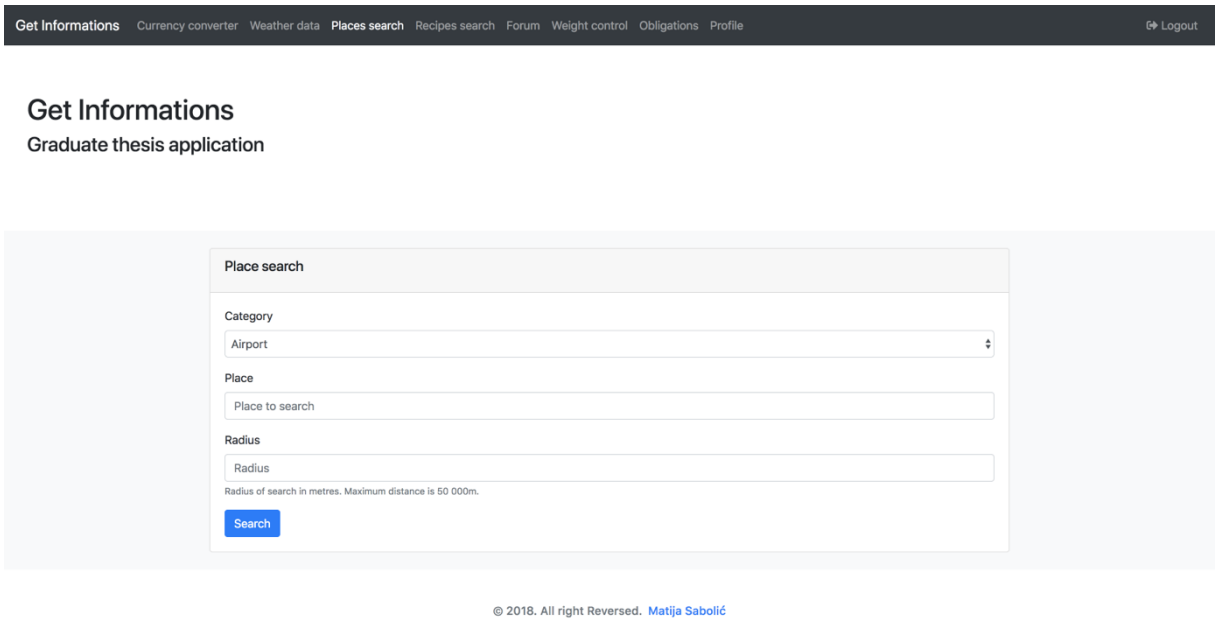
Nakon što korisnik upiše i odabere sve potrebe podatke na početku se poziva REST Web servis za pretvorbu adrese u geolokaciju. Nakon što su dobiveni geolokacijski podaci na temelju odabira iz padajućeg izbornika poziva se REST Web servis za dohvaćanje trenutne vremenske prognoze na temelju zadane geolokacije ili REST Web servis za dohvaćanje petodnevne vremenske prognoze za zadanu geolokaciju. Nakon što REST Web servis vrati odgovarajuće podatke o vremenskoj prognozi oni se prikazuju korisniku na stranici. Na sljedećoj slici prikazan je primjer rezultata ukoliko korisnik zatraži podatke za trenutnu vremensku prognozu.



Slika 24: Prikaz trenutne vremenske prognoze (Izvor: Izrada autora)

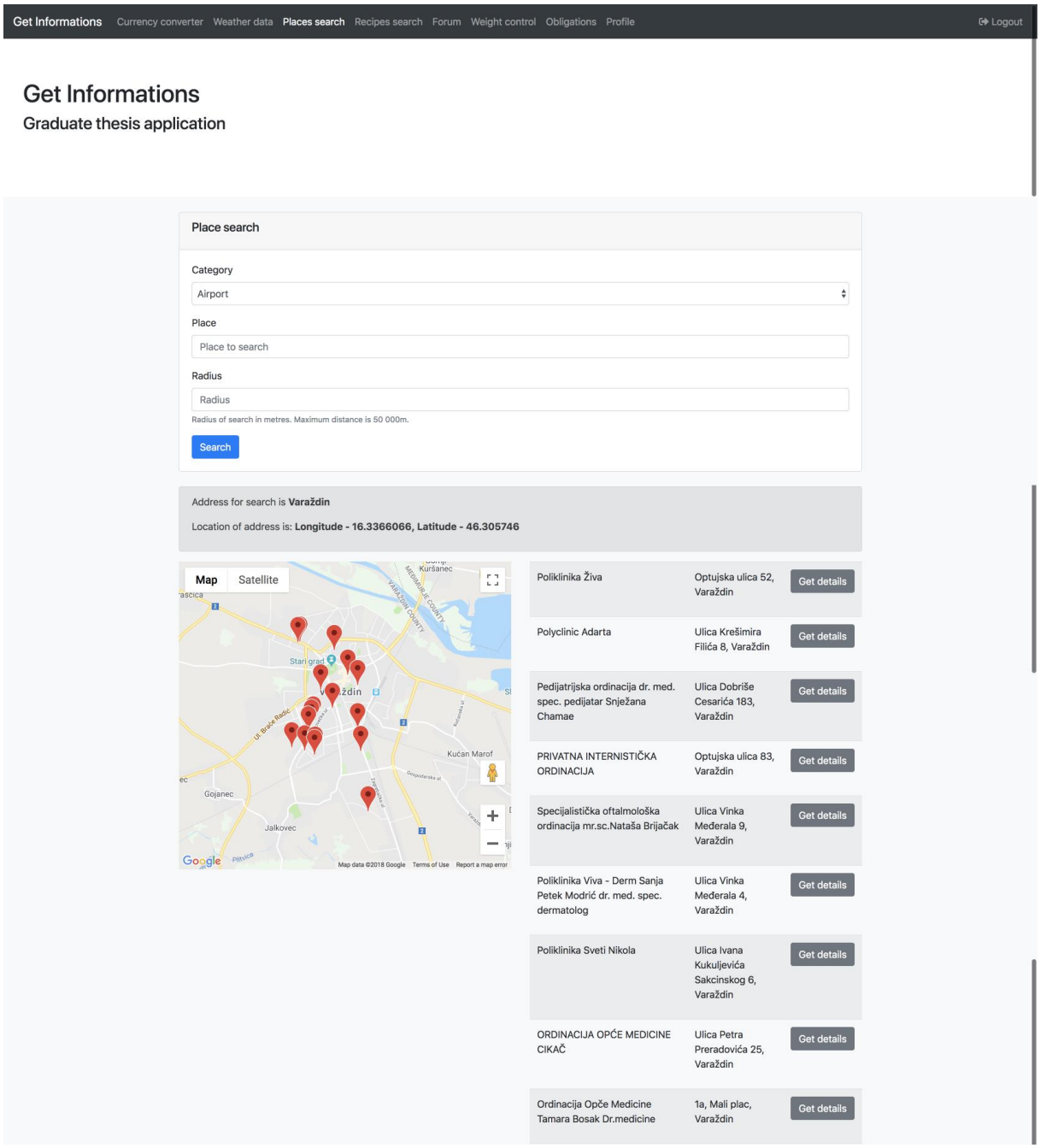
Kako je prikazano na prethodnoj slici korisnik dobiva podatke o vrsti vremena, lokaciji za koju se prikazuju podaci, dobivenoj temperaturi, tlaku zraka, vlažnosti zraka, brzini vjetra i količini kiše ukoliko je vrijeme kišovito. Ukoliko korisnik odabere da želi podatke o vremenskoj prognozi za pet dana dobiva te sve podatke za svaka tri sata.

Sljedeća funkcionalnost sustava je pretraga mjesta. Pretraga se vrši na temelju tipa mjesta, zadane adrese te zadanog radijusa u kojemu se vrši pretraga. Stranica za pretragu mjesta prikazana je na sljedećoj slici.



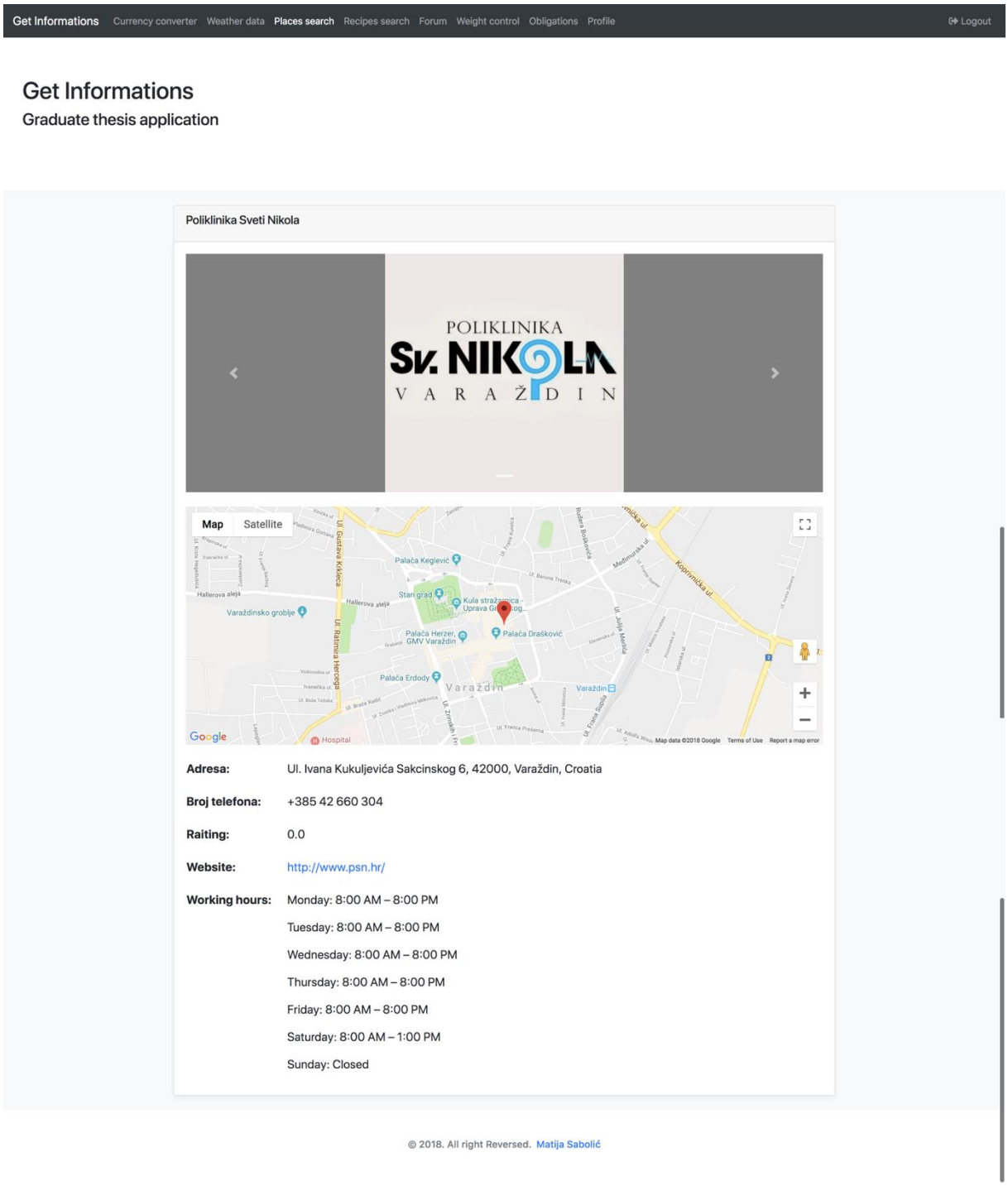
Slika 25: Stranica za pretragu mjesta (Izvor: Izrada autora)

Kada se popuni forma koja je prikazana na prethodnoj slici i korisnik klikne na gumb *Search* na početku se poziva REST Web servis za dobivanje geolokacije na temelju upisane adrese. Kada je poznata geolokacija poziva se REST Web servis za dobivanje mjesta na temelju vrste mjesta, geolokacije te radijusa pretrage. Kada su dobiveni podaci za mjesta oni se prikazuju korisniku na stranici. Također prilikom učitavanja stranice poziva se i *JavaScript API* pomoću kojega se na stranici prikazuje Google mapa sa označenim svim mjestima. Na sljedećoj slici prikazan je rezultat pretrage mjesta.



Slika 26: Prikaz rezultata pretrage mjesta (Izvor: Izrada autora)

Za svako mjesto dobiva se njegovo ime, adresa te link koji vodi korisnika do stranice koja sadrži detaljnije podatke o tom mjestu. Ukoliko postoji više od dvadeset mjesta koja odgovaraju pretrazi ona se prikazuju klikom na gumb *Next* koji se pojavljuje na dnu liste mjesta. Ukoliko korisnik klikne na gumb *Get details* korisniku se prikazuje stranica na kojoj su detaljniji podaci određenom mjestu. Stranica sa detaljnim podacima prikazana je na sljedećoj slici.

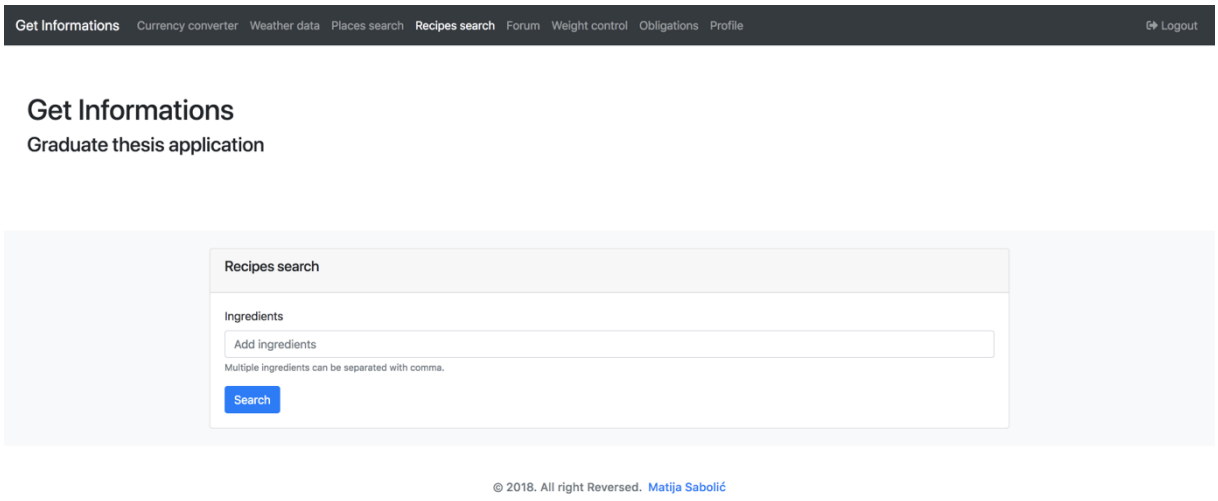


Slika 27: Stranica sa detaljnim podacima za određeno mjesto (Izvor: Izrada autora)

Kada se pozove stranica za dobivanje detaljnih podataka za određeno mjesto poziva te REST Web servis koji vraća veliki skup podataka koje odgovaraju za to mjesto. Zatim se na stranicu korisniku prikazuju lista slika tog mjesta, Google mapa pomoću *JavaScript API-a* sa

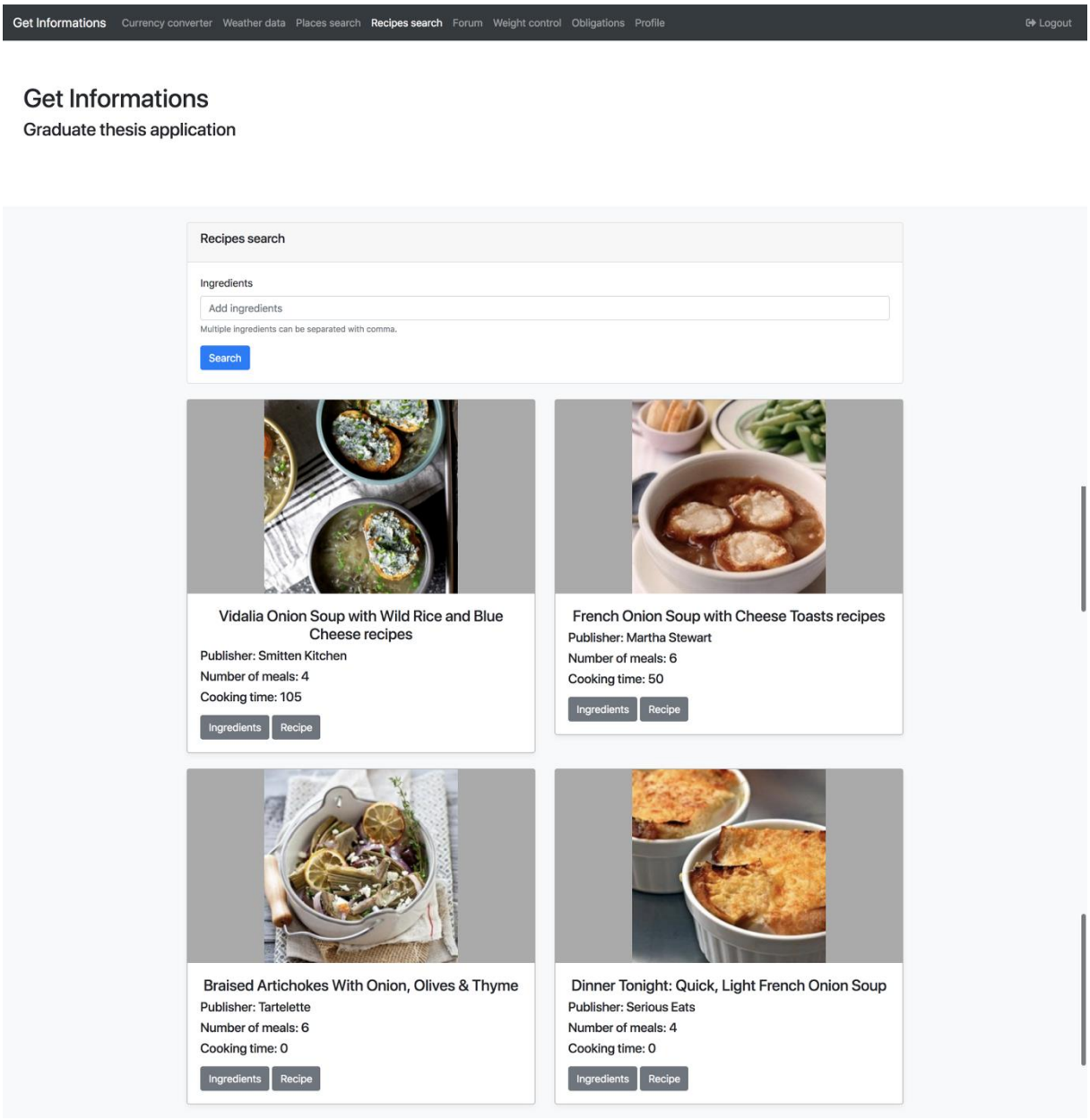
označenom lokacijom mjesta na karti, adresa mjesta, broj telefona, rejting mjesta kojeg ima na Googlu, Web stranica mjesta te radno vrijeme ukoliko ga to mjesto posjeduje.

Sljedeća funkcionalnost aplikacije je pretraga kulinarskih recepata prema željenim sastojcima. Stranica na koju korisnik unosi željene sastojke prikazana je na sljedećoj slici.



Slika 28: Stranica za pretragu kulinarskih recepata (Izvor: Izrada autora)

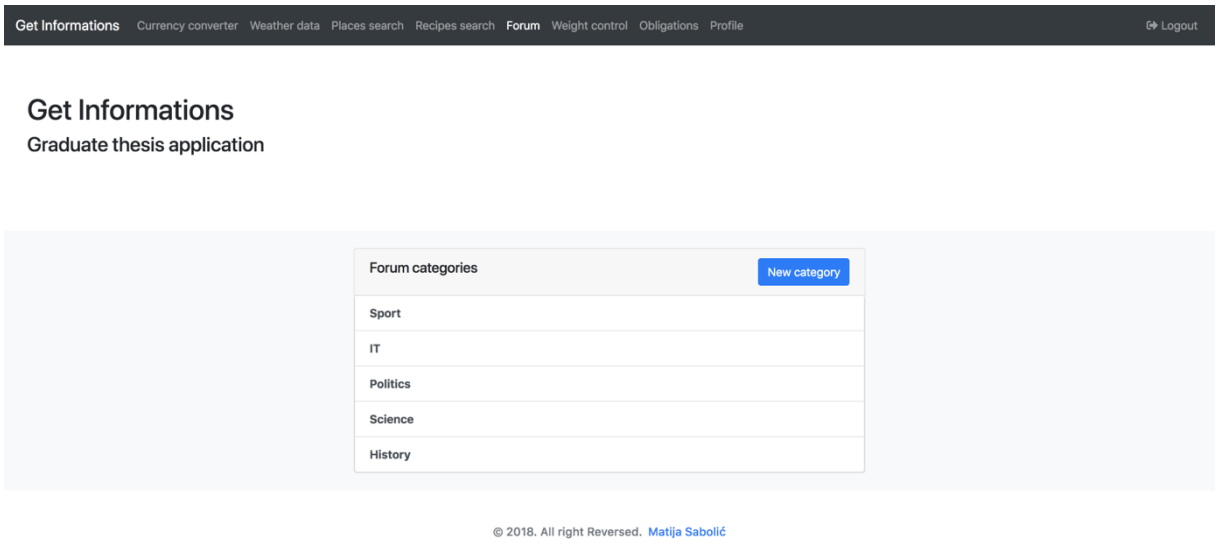
Korisnik u formu koja je prikazana na prethodnoj slici unosi nazive sastojaka odvojene zarezom na engleskom jeziku i klikom na gumb *Search* poziva se REST Web servis koji vraća listu recepata prema zadanim sastojcima. Ti recepti se tada korisniku prikazuju na stranici kako je prikazano na sljedećoj slici.



Slika 29: Prikaz rezultata pretrage kulinarskih recepata (Izvor: Izrada autora)

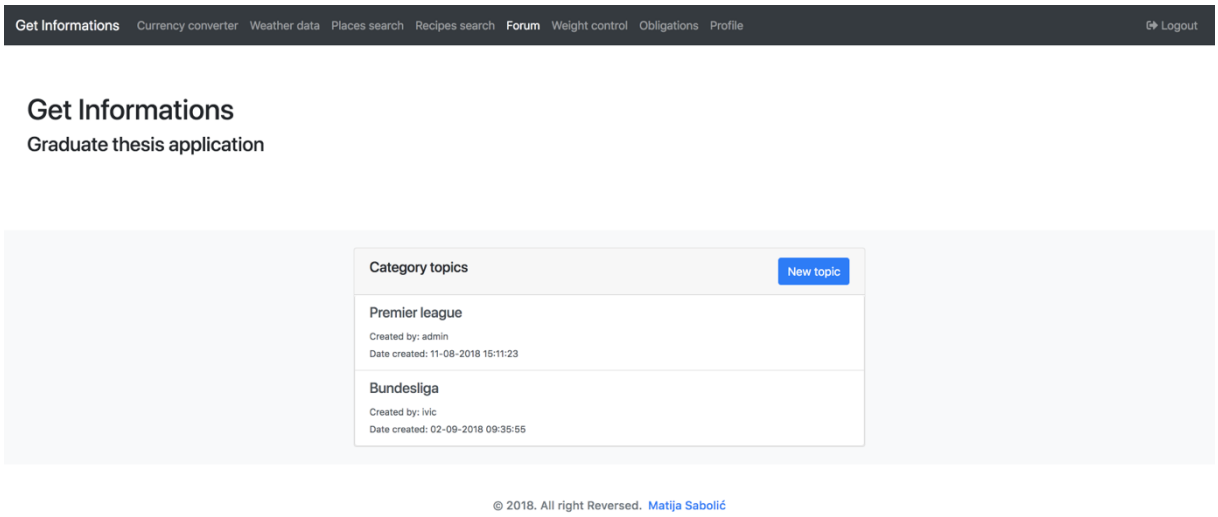
Korisniku ne na stranici prikazuje lista kulinarskih recepata prema njegovim željama. Svaki kulinarski recept sastoji se od njegove slike, naziva, njegovog izdavača, broja koliko porcija sadrži i vremena potrebnog za kuhanje u minutama. Također sadrži gumb *Ingredients* te kada se na njega klikne otvara se modalni prozor sa svim potrebnim sastojcima za izradu tog jela. Također sadrži i gumb *Recipe* te kada se na njega klikne u novom prozoru Web preglednika otvara se originalan recept na Web stranici na kojoj je objavljen.

Sljedeća funkcionalnost aplikacije je forum gdje korisnici mogu razmjenjivati mišljenja o pojedinim temama. Na početku je potrebno izabrati određenu kategoriju foruma ili je moguće kreirati vlastitu kategoriju. Stranica za odabir ili unos nove kategorije prikazana je na sljedećoj slici.



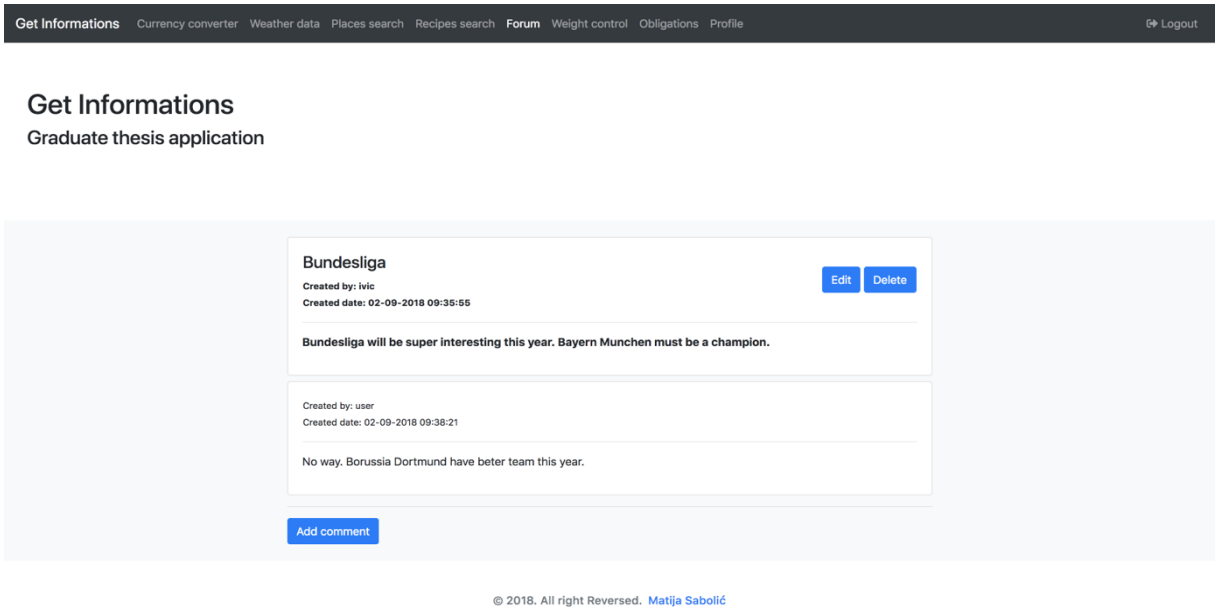
Slika 30: Stranica za odabir kategorije foruma (Izvor: Izrada autora)

Prilikom dolaska na ovu stranicu poziva se REST Web servis koji daje popis svih kategorija koje su dostupne u forumu te se one prikazuju korisniku na stranici. Kada korisnik klikne na gumb *New category* otvara se modalni prozor u koji unosi ime nove kategorije. Tada se poziva REST Web servis koji kreira novu kategoriju u forumu ako ime kategorije već nije zauzeto. REST Web servis vraća korisniku odgovor da je kategorija uspješno napravljena ili poruku greške. Kada korisnik odabere jednu od kategorija iz liste poziva se REST Web servis koji daje listu objava koje se nalaze u toj kategoriji. Stranica s listom objava prikazana je na sljedećoj slici.



Slika 31: Stranica s listom objava određene kategorije foruma (Izvor: Izrada autora)

Na prethodnoj slici prikazana je stranica na kojoj se nalazi lista objava koje su postavljene u određenu kategoriju foruma. Za svaku objavu prikazuje se njezin naslov, autor te vrijeme kreiranja. Ukoliko korisnik klikne na gumb *New topic* otvara mu se modalni prozor gdje upisuje naziv nove objave i sadržaj te nove objave. Kada korisnik unese te sve podatke poziva se REST Web servis koji za nove podatke kreira novu objavu u forumu koja je dostupna svim korisnicima sustava.

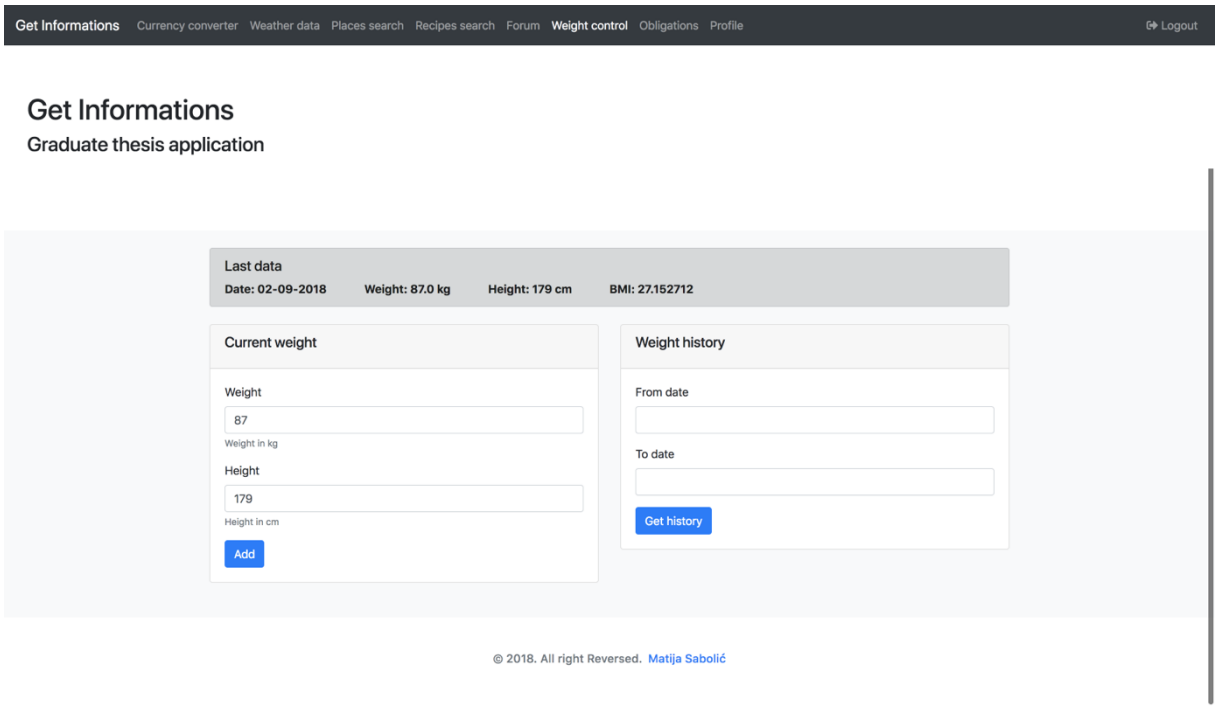


Slika 32: Stranica s detaljima objave foruma (Izvor: Izrada autora)

Klikom na određenu objavu u listi objava poziva se REST Web servis koji za određenu objavu pronalazi njezin cijeli sadržaj zajedno sa svim njezinim komentarima. Zatim se ti podaci prikazuju na stranici s detaljima objave koja je prikazana na prethodnoj slici. Za objavu koja je odabrana vidljivi su podaci o nazivu, autoru, datumu i vremenu kreiranja i sadržaj objave. Ukoliko je prijavljeni korisnik autor objave on ju može obrisati klikom na gumb *Delete* te može i promijeniti njezin sadržaj klikom na gumb *Edit*. Ukoliko korisnik klikne na gumb *Edit* otvara se modalni prozor u kojemu mijenja naziv i sadržaj objave prema svojim željama. Nakon što potvrdi te podatke oni se šalju na REST Web servis koji mijenja sadržaj objave u bazi podataka. Ukoliko korisnik klikne na gumb *Delete* poziva se REST Web servis koji u bazi podataka briše podatke o toj objavi kao i podatke o svim komentarima koji su bili postavljeni na tu objavu.

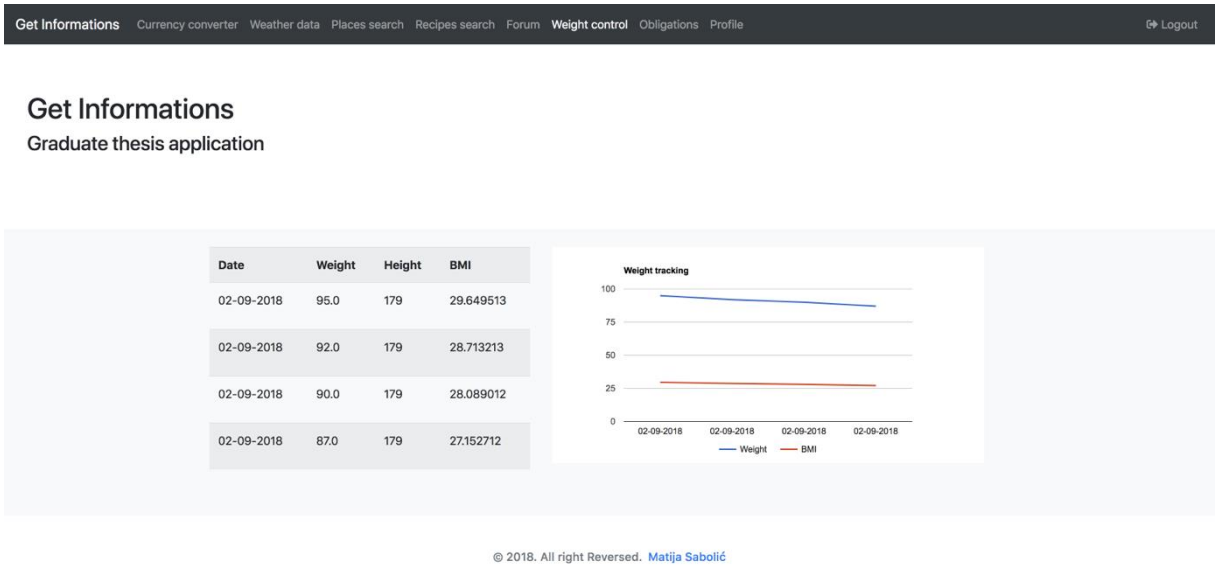
Za svaku objavu na stranici su vidljivi i njezini komentari. Za svaki komentar prikazan je njegov autor, vrijeme kreiranja te njegov sadržaj. Klikom na gumb *New comment* otvara se modalni prozor gdje korisnik može unijeti novi komentar. Kada se potvrdi forma u modalnom prozoru podaci o novom komentaru šalju se na REST Web servis koji sprema podatke o novom komentaru u bazu podataka. Ukoliko je prijavljeni korisnik autor određenog komentara on ima mogućnost istoga obrisati ili promijeniti. Ukoliko korisnik klikne na gumb *Delete* pokraj svojeg komentara poziva se REST Web servis koji u bazi podataka briše podatke za taj komentar. Ukoliko korisnik klikne na gumb *Edit* pokraj svojeg komentara otvara mu se modalni prozor gdje može promijeniti sadržaj komentara. Kada se potvrdi forma u modalnom prozoru poziva se REST Web servis koji u tom slučaju mijenja podatke o tom komentaru u bazi podataka.

Sljedeća funkcionalnost aplikacije je kontrola težine. Klikom na ovu funkcionalnost korisnik je u mogućnosti vidjeti svoje zadnje unesene podatke za težinu i visinu kao i datum kada su podaci uneseni te njegov trenutni BMI indeks. Ti podaci dobivaju se preko SOAP Web servisa koje pruža poslužiteljska aplikacija. Također na stranici je omogućeno i dodavanje novog unosa težine i visine. Kada korisnik klikne na gumb *Add* podaci se šalju SOAP Web servisu koji sprema te nove podatke u bazu podataka. Prethodno opisana stranica prikazana je na sljedećoj slici.



Slika 33: Prikaz stranice za kontrolu težine (Izvor: Izrada autora)

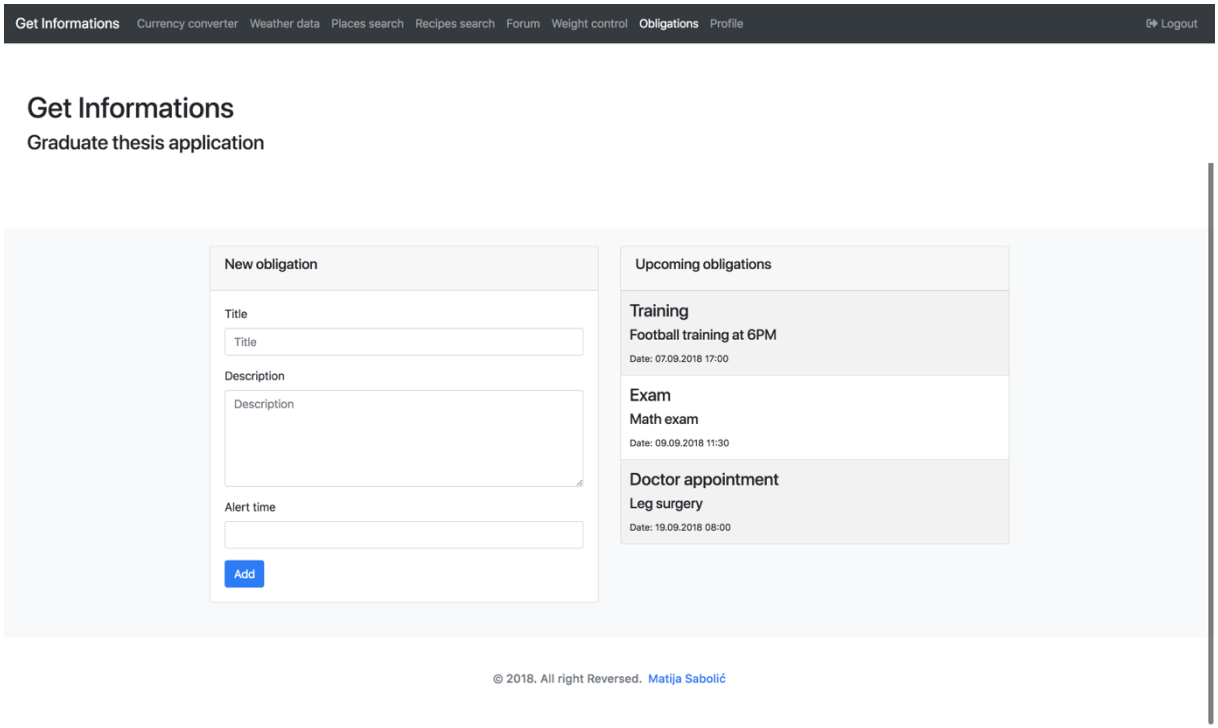
Na desnoj strani stranice nalazi se forma pomoću koje je moguće dobiti unesene podatke u određenom vremenskom periodu kojega zada korisnik. Kada se potvrdi ta forma klikom na gumb *Get* history poziva se SOAP Web servis koji vraća listu unesenih podataka iz baze podataka. Ti podaci se zatim prikazuju na stranici koja je prikazana na sljedećoj slici.



Slika 34: Stranica s podacima o određenom vremenskom periodu (Izvor: Izrada autora)

Na prethodnoj slici prikazani su podaci koje je korisnik unio u određenom vremenskom periodu. Podaci su prikazani također pomoću grafikona kako bi se lakše mogla pratiti oscilacija težine i BMI indeksa.

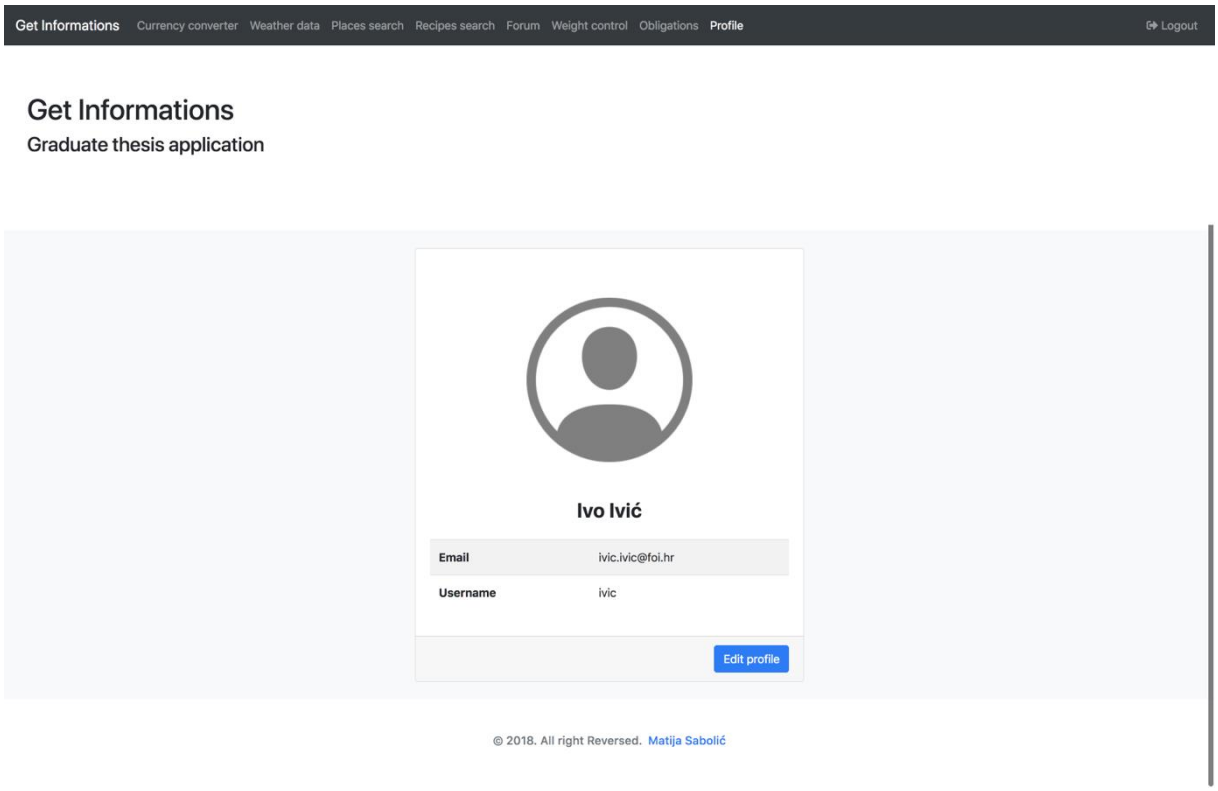
Sljedeća funkcionalnost aplikacije su obaveze korisnika. Prilikom pozivanja ove funkcionalnosti poziva se SOAP Web servis koji daje listu nadolazećih obaveza te se one tada korisniku prikazuju na stranici koja je prikazana na sljedećoj slici.



Slika 35: Prikaz stranice sa obavezama korisnika (Izvor: Izrada autora)

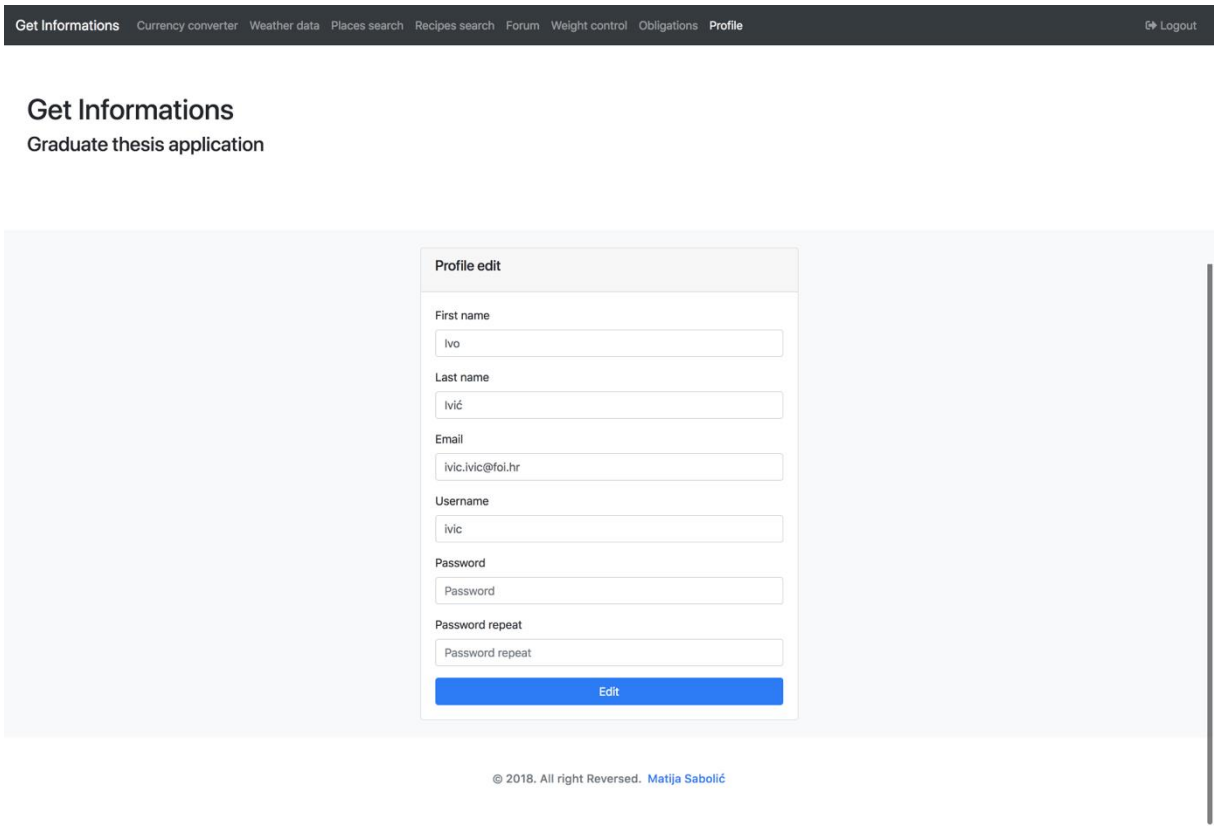
Na lijevoj strani prethodne slike vidljiva je forma pomoću koje korisnik može dodati novu obavezu u svoju listu obaveza. Potrebno je upisati naziv obaveze, opis obaveze te datum kada se treba korisnika obavijestiti o obavezi. Uneseni podaci se klikom na gumb *Add* šalju SOAP Web servisu koji tada te podatke sprema u bazu podataka. Obavještavanje korisnika o nadolazećoj obavezi se radi putem poslužiteljske aplikacije koja pomoću email poslužitelja šalje korisniku email poruku 10 minuta prije što je njegova obaveza zakazana.

Sljedeća funkcionalnost aplikacije je prikaz profila korisnika. Odabirom ove funkcionalnosti putem REST Web servisa dobivaju se podaci o prijavljenom korisniku te se isti prikazuju na stranici koja je prikazana na sljedećoj slici.



Slika 36: Prikaz korisničkog profila (Izvor: Izrada autora)

Korisnički profil sadrži podatke koji su u bazi podataka spremljeni za prijavljenog korisnika. Prikazani su podaci o imenu i prezimenu, email adresi te korisničkom imenu. Na stranici se nalazi i gumb *Edit profile* te kada se klikne na njega otvara se stranica za promjenu korisničkih podataka koja je prikazana na sljedećoj slici.



Slika 37: Stranica za promjenu korisničkih podataka (Izvor: Izrada autora)

Odabirom funkcionalnosti promjena korisničkih podataka poziva se REST Web servis koji daje podatke o prijavljenom korisniku te se oni prikazuju u formi. Korisnik može svoje podatke promijeniti te klikom na gumb *Edit* ti podaci se šalju na REST Web servis koji na početku provjerava da li uneseni podaci nisu već spremljeni za nekog drugog korisnika te ukoliko nisu podaci se spremaju u bazu podataka i REST Web servis vraća odgovor korisniku kako su njegovi podaci uspješno promijenjeni. Ukoliko se uneseni podaci podudaraju sa podacima nekog drugog korisnika sustava REST Web servis vraća poruku greške te se ona prikazuje korisniku na stranici.

8.5. Opis implementacije sustava

Kako su u svrhu diplomskog rada implementirane klijentska i poslužiteljska aplikacija potrebno je opisati na koji način je obavljena njihova implementacija. Kako se radi o dvije zasebne aplikacije na početku će biti opisani glavni dijelovi implementacije poslužiteljske aplikacije a zatim će biti opisani glavni dijelovi implementacije klijentske aplikacije.

8.5.1. Opis implementacije poslužiteljske aplikacije

Kao što je već navedeno u prethodnim poglavljima poslužiteljska aplikacije implementirana je u Java programskom jeziku te Spring programskom okviru. Uzorak dizajna koji je korišten kod njezine izrade je MVC. Svaki MVC uzorak dizajna sastoji se od modela, pogleda i kontrolera.

U poslužiteljskoj aplikaciji svaki model predstavlja jednu Java klasu koja reprezentira jednu tablicu u PosgreSQL bazi podataka. Primjer klase modela naveden je u nastavku.

```
1.  package com.unizg.foi.GetInformationsServices.model.database;
2.
3.  import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
4.
5.  import javax.persistence.*;
6.
7.  @Entity
8.  @Table(name = "user", schema = "get_informations")
9.  @JsonIgnoreProperties(value = "password")
10. public class User {
11.
12.     @Id
13.     @GeneratedValue(strategy=GenerationType.IDENTITY)
14.     @Column(name="id")
15.     private int id;
16.
17.     @Column(name = "username")
18.     private String username;
19.
20.     @Column(name = "first_name")
21.     private String firstName;
22.
23.     @Column(name = "last_name")
24.     private String lastName;
25.
26.     @Column(name = "email")
27.     private String email;
28.
29.     @Column(name = "password")
30.     private String password;
31. }
```

Prethodni primjer klase modela predstavlja tablicu korisnika u bazi podataka. Anotacija *@entity* označava da ta jedna instance ove klase predstavlja jedan entitet korisnika iz baze podataka. Anotacija *@Table(name = "user", schema = "get_informations")* označava da se entiteti korisnika nalaze u tablici *user* koja ima shemu *get_informations*. Anotacija *@JsonIgnoreProperties(value = „password“)* označava da se vrijednost lozinke neće prikazivati kod rezultata REST Web servisa.

Varijable koje ispred svoje deklaracije imaju navedenu anotaciju *@column* predstavljaju stupce tablice u bazi podataka. Varijabla koja ispred svoje deklaracije ima navedene još i anotacije *@Id* i *@GeneratedValue* predstavlja stupac u tablici koji je primarni ključ te se njegova vrijednost automatski generira.

Kako svaka tablica u bazi podataka ima odgovarajuću klasu modela tako ima i odgovarajuću klasu kontrolera. U poslužiteljskoj aplikaciji kontroler služi kao klasa u kojoj su definirana mapiranja za operacije REST Web servisa. Jedan jednostavan primjer kontrolera prikazan je u nastavku.

```
1.  package com.unizg.foi.GetInformationsServices.controller;
2.
3.  import com.unizg.foi.GetInformationsServices.model.Response;
4.  import com.unizg.foi.GetInformationsServices.model.database.User;
5.  import com.unizg.foi.GetInformationsServices.services.UserService;
6.  import org.springframework.beans.factory.annotation.Autowired;
7.  import org.springframework.util.ObjectUtils;
8.  import org.springframework.web.bind.annotation.*;
9.
10. import java.util.ArrayList;
11. import java.util.List;
12.
13. @RestController
14. public class UserController {
15.
16.     @Autowired
17.     private UserService userService;
18.
19.     @RequestMapping(value = "/rest/user", method = RequestMethod.GET)
20.     public Response userData(@RequestParam("username") String username)
21.     {
22.         Response response = new Response();
23.
24.         User user = userService.findByUsername(username);
25.
26.         if(!ObjectUtils.isEmpty(user)){
27.             response.setStatus("OK");
28.             response.setError(StringUtils.EMPTY);
29.             response.setResponse(user);
30.         } else {
31.             response.setStatus("ERROR");
32.             response.setError("User not found");
33.             response.setResponse(StringUtils.EMPTY);
34.         }
35.
36.         return response;
37.     }
38. }
```

U prethodnom primjeru anotacija *@RestController* označava da metode kontrolera služe kao krajnje točke za REST Web servis i da se svi objekti serijaliziraju u JSON format zapisa prilikom njihovog ispisa.

Anotacija *@RequstMapping* specificira krajnju točku REST Web servisa zajedno sa HTTP metodom kojom se mora napraviti zahtjev kako bi se pozvala metoda koje je definirana ovom anotacijom. Anotacija *@RequestParam* označava GET parametar u zahtjevu koji mora biti specificiran kako bi ta metoda mogla biti pozvana.

Pomoću anotacije *@Autowired* omogućeno je korištenje servisa i njegovih metoda. Servisi će biti opisani kasnije u ovom poglavlju.

Kod poslužiteljske aplikacije pogledi nisu striktno definirani iz razloga što se objekti iz kontrolera serijaliziraju te su u tom slučaju na pogledu korisniku prikazani ti serijalizirani podaci. U primjeru kontrolera koji je prethodno naveden nakon što se odradi sva logika koja definirana u metodi objekt *response* se serijalizira te se ti serijalizirani podaci ispisuju korisnika u obliku pogleda.

Komunikacija s bazom podataka implementirana je pomoću klasa koje predstavljaju repozitorije. Primjer repozitorija naveden je u nastavku.

```
1.  package com.unizg.foi.GetInformationsServices.repository;
2.
3.  import com.unizg.foi.GetInformationsServices.model.database.User;
4.  import org.springframework.data.jpa.repository.JpaRepository;
5.  import org.springframework.stereotype.Repository;
6.
7.  @Repository
8.  public interface UserRepository extends JpaRepository<User, Integer> {
9.
10.     User findById(int ID);
11.     User findByUsername(String username);
12.     User findByEmail(String email);
13.     User findByUsernameAndPassword(String username, String password);
14.
15. }
```

Anotacija *@Repository* označava da klasa koja ima direktnu komunikaciju s bazom podataka. Upiti te druge operacije nad bazom provode se preko jezika upita koji je definiran u sučelju *JpaRepositroy*. Tako metoda *findById(int ID)* predstavlja upit nad bazom podataka koji vraća korisnika sa zadanim identifikatorom.

Sljedeći primjer predstavlja servis u Spring programskom okviru. Servis označava klasu koja obavlja neku uslugu, kao što je izvršavanje poslovne logike, izvršavanje izračuna i pozivanje eksternih API-a. Servis se specificira anotacijom *@Service* te svojim nazivom u zagradama kao u sljedećem primjeru.

```java
package com.unizg.foi.GetInformationsServices.services.impl;

import com.unizg.foi.GetInformationsServices.model.database.User;
import com.unizg.foi.GetInformationsServices.repository.UserRepository;
import com.unizg.foi.GetInformationsServices.services.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service("userService")
public class UserServiceImpl implements UserService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public User save(User user) {
        return userRepository.save(user);
    }

    @Override
    public User updateUser(User user) {
        return userRepository.save(user);
    }

    @Override
    public List<User> findAll() {
        return userRepository.findAll();
    }

    @Override
    public User findById(int id) {
        return userRepository.findById(id);
    }

    @Override
    public User findByUsername(String username) {
        return userRepository.findByUsername(username);
    }

    @Override
    public User findByEmail(String email) {
        return userRepository.findByEmail(email);
    }

    @Override
    public User findByUsernameAndPassword(String username, String password) {
        return userRepository.findByUsernameAndPassword(username, password);
    }
}
```

U prethodnom primjeru pomoću anotacije *@Autowired* omogućeno je korištenje metoda koje pruža repozitorij. Prethodni servis pruža metode pomoću kojih je također omogućen pristup bazi podataka pomoću metoda iz repozitorija.

Poslužiteljska aplikacija također ima za zadatak slanje email poruka korisnicima. Primjer klase koja se bavi slanjem email poruka korisnicima naveden je u nastavku.

```
1.  package com.unizg.foi.GetInformationsServices.utils;
2.
3.  import com.unizg.foi.GetInformationsServices.model.database.Obligation;
4.  import com.unizg.foi.GetInformationsServices.services.EmailSendService;
5.  import com.unizg.foi.GetInformationsServices.services.ObligationService;
6.  import org.springframework.beans.factory.annotation.Autowired;
7.  import org.springframework.scheduling.annotation.Scheduled;
8.  import org.springframework.stereotype.Component;
9.
10. import java.time.LocalDateTime;
11. import java.util.List;
12.
13. @Component
14. public class EmailSender {
15.
16.     @Autowired
17.     private EmailSendService emailSendService;
18.
19.     @Autowired
20.     private ObligationService obligationService;
21.
22.     @Scheduled(cron = "0 */2 * * * ?")
23.     public void sendNotification() {
24.         LocalDateTime fromDate = LocalDateTime.now().plusMinutes(9);
25.         LocalDateTime toDate = LocalDateTime.now().plusMinutes(10);
26.
27.         List<Obligation> list = obligationService.findByAlertTimeInRange(fromDate, t
    oDate);
28.
29.         for(Obligation o : list) {
30.             String emailBody = "<p>Hello " + o.getUser().getUsername() + "</p>";
31.             emailBody = emailBody + "<p>We just want to tell you that you have oblig
    ation in " + o.getAlertTimeString() + "</p>";
32.             emailBody = emailBody + "<p>Obligation title: " + o.getTitle() + "</p>";

33.             emailBody = emailBody + "<p>Obligation description: " + o.getDescription
    () + "</p>";
34.
35.             emailSendService.sendEmail(o.getTitle(), emailBody, o.getUser().getEmail
    ());
36.         }
37.     }
38.
39. }
```

Anotacija *@Scheduled(cron = "0 */2 * * * ?")* u prethodnom primjeru označava da se metoda koja je ovako anotirana poziva periodički svakih dvije minute. U prethodnom primjeru svake dvije minute šalju se email poruke odgovarajućim korisnicima kako bi ih se podsjetilo kako ubrzo imaju neku obavezu.

Što se tiče poslužiteljske aplikacije potrebno je još opisati implementaciju SOAP Web servisa. Sve metode SOAP Web servisa definiraju se pomoću XML sheme koju će Spring automatski pretvoriti u WSDL datoteku. Primjer XML sheme naveden je u nastavku.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:tns="http://www.GetInformationsServices.foi.unizg.com/soap"
      targetNamespace="http://www.GetInformationsServices.foi.unizg.com/soap"
      elementFormDefault="qualified">

    <xs:element name="addObligationRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="username" type="xs:string"/>
                <xs:element name="title" type="xs:string"/>
                <xs:element name="descriptions" type="xs:string"/>
                <xs:element name="alertTime" type="xs:dateTime"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="addObligationResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="statusCode" type="xs:string"/>
                <xs:element name="statusMessage" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="getObligationsByUserRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="username" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="getObligationsByUserResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="statusCode" type="xs:string"/>
                <xs:element name="statusMessage" type="xs:string"/>
                <xs:element name="obligation" type="tns:obligation"
                            maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="obligation">
        <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="description" type="xs:string"/>
            <xs:element name="alertTime" type="xs:dateTime"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Nakon što je su definirane sve metode i tipovi podataka unutar XML sheme potrebno je generirati Java klase koje predstavljaju ove metode i tipove podataka. To se u Springu obavlja automatski pomoću Maven dodatka koji se dodaje u pom.xml datoteku. Maven dodatak prikazan je u sljedećem primjeru.

```
<plugin>
   <groupId>org.codehaus.mojo</groupId>
   <artifactId>jaxb2-maven-plugin</artifactId>
   <version>1.6</version>
   <executions>
      <execution>
         <id>xjc</id>
         <goals>
            <goal>xjc</goal>
         </goals>
      </execution>
   </executions>
   <configuration>
      <schemaDirectory>${project.basedir}/src/main/resources/xsds/</schemaDirectory>
      <outputDirectory>${project.basedir}/src/main/java</outputDirectory>
      <clearOutputDir>false</clearOutputDir>
   </configuration>
</plugin>
```

Pomoću ovog dodatka biti će prepoznate sve XML sheme koje su smještene na putanji *src/main/resources/xsds/* te će iz njih biti generirane sve potrebne Java klase.

Nakon što su generirane sve Java klase iz XML sheme potrebno je napraviti krajnje točke SOAP Web servisa. Primjer klase koja definira krajnje točke nalazi se u nastavku.

```
package com.unizg.foi.GetInformationsServices.soap.endpoints;

import com.unizg.foi.GetInformationsServices.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.util.ObjectUtils;
import org.springframework.ws.server.endpoint.annotation.*;

import javax.xml.datatype.DatatypeConfigurationException;
import java.time.LocalDateTime;
import java.util.List;

@Endpoint
public class ObligationEndpoint {

    private static final String NAMESPACE_URI = "http://www.GetInformationsServices.
foi.unizg.com/soap";

    @Autowired
    private ObligationService obligationService;

    @Autowired
    private UserService userService;

    @PayloadRoot(namespace = NAMESPACE_URI, localPart = "addObligationRequest")
```

```
24.     @ResponsePayload
25.     public AddObligationResponse addObligation(@RequestPayload AddObligationRequest
    request) {
26.         AddObligationResponse response = new AddObligationResponse();
27.
28.         User user = userService.findByUsername(request.getUsername());
29.
30.         if(!ObjectUtils.isEmpty(user)) {
31.             Obligation obligation = new Obligation();
32.             obligation.setUser(user);
33.             obligation.setTitle(request.getTitle());
34.             obligation.setDescription(request.getDescriptions());
35.             obligation.setAlertTime(DateConverter.convertToLocalDateTime(request.get
    AlertTime()));
36.             obligationService.save(obligation);
37.
38.             response.setStatusCode("OK");
39.         } else {
40.             response.setStatusCode("ERROR");
41.             response.setStatusMessage("User not found");
42.         }
43.
44.         return response;
45.     }
46.
47. }
```

Anotacija *@Endpoint* označava kako klasa s ovom anotacijom može služiti za obradu SOAP poruka. Anotacija *@PayloadRoot* označava metodu koja će biti pozvana za određeno područje imena u SOAP poruci i za određenu SOAP operaciju. Anotacija *@RequestPayload* označuje da se sadržaj određene SOAP poruke biti mapiran u varijablu *request*. Anotacija *@ResponsePayload* označava da će povratna vrijednost SOAP Web servisa biti mapirana u SOAP poruku.

Kada su implementirane sve krajnje točke SOAP Web servisa potrebno je još implementirati konfiguraciju SOAP Web servisa kako bi sve moglo raditi kako treba. Klasa koja predstavlja konfiguraciju SOAP Web servisa nalazi se u nastavku.

```
1.  package com.unizg.foi.GetInformationsServices.config;
2.
3.  import org.springframework.boot.web.servlet.ServletRegistrationBean;
4.  import org.springframework.context.ApplicationContext;
5.  import org.springframework.context.annotation.Bean;
6.  import org.springframework.context.annotation.Configuration;
7.  import org.springframework.core.io.ClassPathResource;
8.  import org.springframework.ws.config.annotation.EnableWs;
9.  import org.springframework.ws.config.annotation.WsConfigurerAdapter;
10. import org.springframework.ws.transport.http.MessageDispatcherServlet;
11. import org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition;
12. import org.springframework.xml.xsd.SimpleXsdSchema;
13. import org.springframework.xml.xsd.XsdSchema;
14.
15. @EnableWs
```

```
16. @Configuration
17. public class WebServiceConfig extends WsConfigurerAdapter {
18. 
19.     @Bean
20.     public ServletRegistrationBean messageDispatcherServlet(ApplicationContext appli
    cationContext) {
21.         MessageDispatcherServlet servlet = new MessageDispatcherServlet();
22.         servlet.setApplicationContext(applicationContext);
23.         servlet.setTransformWsdlLocations(true);
24.         return new ServletRegistrationBean(servlet, "/soap/*");
25.     }
26. 
27.     @Bean(name = "obligations")
28.     public DefaultWsdl11Definition defaultObligationsWsdl11Definition(XsdSchema obli
    gationsSchema) {
29.         DefaultWsdl11Definition wsdl11Definition = new DefaultWsdl11Definition();
30.         wsdl11Definition.setPortTypeName("ObligationsPort");
31.         wsdl11Definition.setLocationUri("/soap/obligations");
32.         wsdl11Definition.setTargetNamespace("http://www.GetInformationsServices.foi.
    unizg.com/soap");
33.         wsdl11Definition.setSchema(obligationsSchema);
34.         return wsdl11Definition;
35.     }
36. 
37.     @Bean
38.     public XsdSchema obligationsSchema() {
39.         return new SimpleXsdSchema(new ClassPathResource("xsds/obligation.xsd"));
40.     }
41. 
42. }
```

Anotacija *@EnableWs* zajedno sa anotacijom *@Configuration* koristi se kako bi SOAP Web servis bio dostupan u klasi *WsConfigurerAdapter*. U klasi *ServletRegistrationBean* konfigurirani je aplikacijski kontekst, URL mapiranja i drugo. Klasa *DefaultWsdl11Definition* konfigurira WSDL definicije kao što su ime tipa porta, URI, ciljano područje imena, shema i ostalo. Klasa *XsdSchema* predstavlja apstrakciju za XSD shemu. Nakon što se implementira konfiguracija zajedno sa svim prije navedenim dijelovima dobiva se potpuno funkcionalan SOAP Web servis.

8.5.2. Opis implementacije klijentske aplikacije

Kao i poslužiteljska aplikacija, klijentska aplikacija implementirana je u Java programskom jeziku te Spring programskom okviru. Uzorak dizajna koji je korišten kod njezine izrade također je MVC.

Kod klijentske aplikacije model klase predstavljaju klase koje samo spremaju podatke koji se naknadno šalju na poglede. Model klase ne sadrže logiku, te imaju samo getere i setere pomoću kojih se pristupa varijablama klase. Primjer model klase naveden je u nastavku.

```java
package com.unizg.foi.GetInformations.model;

public class PlaceModel {

    private String placeId;
    private String name;
    private String address;
    private LocationModel location;

    public PlaceModel() {
    }

    public String getPlaceId() {
        return placeId;
    }

    public void setPlaceId(String placeId) {
        this.placeId = placeId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public LocationModel getLocation() {
        return location;
    }

    public void setLocation(LocationModel location) {
        this.location = location;
    }
}
```

Kako kod poslužiteljske aplikacije postoje kontroleri tako oni postoje i kod klijentske aplikacije samo što im je malo drugačija uloga. Primjer jednog kontrolera u klijentskoj aplikaciji prikazan je u nastavku.

```java
package com.unizg.foi.GetInformations.controller;

import com.unizg.foi.GetInformations.model.WeatherModel;
import com.unizg.foi.GetInformations.model.LocationModel;
import com.unizg.foi.GetInformations.services.LocationService;
import com.unizg.foi.GetInformations.services.WeatherService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.util.ObjectUtils;
```

```
11. import org.springframework.web.bind.annotation.RequestMapping;
12. import org.springframework.web.bind.annotation.RequestMethod;
13. import org.thymeleaf.util.StringUtils;
14. 
15. import javax.servlet.http.HttpServletRequest;
16. import java.io.IOException;
17. import java.util.List;
18. 
19. @Controller
20. public class WeatherController {
21. 
22.     @Autowired
23.     private WeatherService weatherService;
24. 
25.     @Autowired
26.     private LocationService locationService;
27. 
28.     @RequestMapping(value = "/weather", method = RequestMethod.GET)
29.     public String weatherPage(Model model)
30.     {
31.         return "weather";
32.     }
33. 
34.     @RequestMapping(value = "/weather", method = RequestMethod.POST)
35.     public String weatherDataPage(HttpServletRequest request, Model model) throws IO
    Exception
36.     {
37.         LocationModel locationModel = locationService.getLocationByAddress(request.g
    etParameter("address"));
38.         if(ObjectUtils.isEmpty(locationModel.getLatitude())){
39.             model.addAttribute("error", "Address not found!");
40.         } else {
41.             model.addAttribute("address", request.getParameter("address"));
42.             model.addAttribute("location", locationModel);
43. 
44.             if(StringUtils.equals(request.getParameter("type"), "current"))
45.             {
46.                 WeatherModel weatherModel = weatherService.getCurrentWeather(locatio
    nModel);
47.                 model.addAttribute("current", weatherModel);
48.             }
49.             else
50.             {
51.                 List<WeatherModel> weatherList = weatherService.getFiveDaysWeather(l
    ocationModel);
52.                 model.addAttribute("list", weatherList);
53.             }
54.         }
55. 
56.         return "weather";
57.     }
58. }
```

Anotacija *@Controller* označava da se radi o kontroleru u MVC uzorku dizajna. Anotacije *@Autowired* su korištene kao i kod poslužiteljske aplikacije kako bi se omogućilo korištenje servisa. Anotacija *@RequestMapping* označava za koji URI i koju HTTP metodu se poziva određena metoda iz kontrolera. Podaci se na pogled šalju pomoću klase *Model*. Povratni

parametar svake od metoda označava naziv pogleda koji će se prikazati korisniku, tako će u ovom slučaju za obje metode biti pozvan pogled *weather.html*.

U klijentskoj aplikaciji se također koriste servisi ali je njihova uloga drugačija nego što je bila kod poslužiteljske aplikacije. U klijentskoj aplikaciji servisi obavljaju pozivanje Web servisa te obavljaju pretvorbu dobivenih rezultata u model klase. Jedan primjer servisa u klijentskoj aplikaciji prikazan je u nastavku.

```
1.  package com.unizg.foi.GetInformations.services.impl;
2.
3.  import com.unizg.foi.GetInformations.constants.ApiKeys;
4.  import org.json.JSONObject;
5.  import org.springframework.stereotype.Service;
6.  import org.springframework.web.client.RestTemplate;
7.  import org.springframework.web.util.UriComponentsBuilder;
8.  import org.thymeleaf.util.StringUtils;
9.
10. import com.unizg.foi.GetInformations.model.LocationModel;
11. import com.unizg.foi.GetInformations.services.LocationService;
12.
13. import java.io.UnsupportedEncodingException;
14.
15. @Service("location")
16. public class LocationServiceImpl implements LocationService
17. {
18.     private static final String GOOGLE_API_BASE_URL = "https://maps.googleapis.com/m
    aps/api/geocode/json";
19.     private static final String KEY_PARAM = "key";
20.     private static final String ADDRESS_PARAM = "address";
21.
22.     private static final String RESULTS = "results";
23.     private static final String GEOMETRY = "geometry";
24.     private static final String LOCATION = "location";
25.     private static final String STATUS = "status";
26.     private static final String STATUS_OK = "OK";
27.     private static final String LATITUDE = "lat";
28.     private static final String LONGITUDE = "lng";
29.
30.     @Override
31.     public LocationModel getLocationByAddress(String address) throws UnsupportedEnco
    dingException
32.     {
33.         LocationModel location = new LocationModel();
34.
35.         UriComponentsBuilder builder = UriComponentsBuilder
36.             .fromUriString(GOOGLE_API_BASE_URL)
37.             .queryParam(KEY_PARAM, ApiKeys.GOOGLE_API_KEY)
38.             .queryParam(ADDRESS_PARAM, address);
39.
40.         RestTemplate request = new RestTemplate();
41.         String result = request.getForObject(builder.build(false).toUriString(), Str
    ing.class);
42.
43.         JSONObject jsonObject = new JSONObject(result);
44.
45.         if(StringUtils.equals(jsonObject.getString(STATUS), STATUS_OK)){
```

```
            location.setLatitude(jsonObject.getJSONArray(RESULTS).getJSONObject(0).getJSONObject(GEOMETRY).getJSONObject(LOCATION).getBigDecimal(LATITUDE));
            location.setLongitude(jsonObject.getJSONArray(RESULTS).getJSONObject(0).getJSONObject(GEOMETRY).getJSONObject(LOCATION).getBigDecimal(LONGITUDE));
        }

        return location;
    }
}
```

Kako bi klijentska aplikacija bila u mogućnosti koristiti SOAP Web servis koji je implementiran pomoću poslužiteljske aplikacije potrebe su joj iste klase koje koristit i poslužiteljska aplikacija. Te klase se u klijentskoj aplikaciji preuzimaju putem Maven dodatka koji traži WSDL datoteku na zadanom URI-u i na temelju njega stvara sve potrebne klase u poslužiteljskoj aplikaciji. Primjer Maven dodatka koji se dodaje u pom.xml datoteku klijentske aplikacije nalazi se u nastavku.

```
<plugin>
   <groupId>org.jvnet.jaxb2.maven2</groupId>
   <artifactId>maven-jaxb2-plugin</artifactId>
   <version>0.13.1</version>
   <executions>
      <execution>
         <goals>
            <goal>generate</goal>
         </goals>
      </execution>
   </executions>
   <configuration>
      <schemaLanguage>WSDL</schemaLanguage>
      <generatePackage>com.unizg.foi.GetInformationsServices.soap</generatePackage>
      <schemas>
         <schema>
            <url>http://get-information-services.herokuapp.com/soap/weights.wsdl</url>
         </schema>
         <schema>
            <url>http://get-information-services.herokuapp.com/soap/obligations.wsdl</url>
         </schema>
      </schemas>
   </configuration>
</plugin>
```

Kada su dobivene sve klase koje koristi i poslužiteljska aplikacija potrebno je implementirati klijenta SOAP Web servisa koji će raditi zahtjeve na SOAP Web servis poslužiteljske aplikacije. Primjer klase koja predstavlja SOAP Web servis klijenta nalazi se u nastavku.

```
package com.unizg.foi.GetInformations.soap;

import com.unizg.foi.GetInformationsServices.soap.*;
import com.unizg.foi.GetInformations.utils.DateConverter;
import org.springframework.ws.client.core.support.WebServiceGatewaySupport;
```

```
6.  import org.springframework.ws.soap.client.core.SoapActionCallback;
7.
8.  import javax.xml.datatype.DatatypeConfigurationException;
9.  import java.time.LocalDateTime;
10.
11. public class ObligationClient extends WebServiceGatewaySupport {
12.
13.     public AddObligationResponse addObligation(String user, String title, String des
    cription, LocalDateTime alertTime) throws DatatypeConfigurationException {
14.         AddObligationRequest request = new AddObligationRequest();
15.         request.setUsername(user);
16.         request.setTitle(title);
17.         request.setDescriptions(description);
18.         request.setAlertTime(DateConverter.convertToXMLDate(alertTime));
19.         AddObligationResponse response = (AddObligationResponse) getWebServiceTempla
    te().marshalSendAndReceive(
20.                 "http://get-information-
    services.herokuapp.com/soap/obligations.wsdl",
21.                 request, new SoapActionCallback("http://get-information-
    services.herokuapp.com/soap/addObligationRequest")
22.         );
23.
24.         return response;
25.     }
26.
27.     public GetObligationsByUserResponse getObligations(String user) {
28.         GetObligationsByUserRequest request = new GetObligationsByUserRequest();
29.         request.setUsername(user);
30.
31.         GetObligationsByUserResponse response = (GetObligationsByUserResponse) getWe
    bServiceTemplate().marshalSendAndReceive(
32.                 "http://get-information-
    services.herokuapp.com/soap/obligations.wsdl",
33.                 request, new SoapActionCallback("http://get-information-
    services.herokuapp.com/soap/getObligationsByUserRequest")
34.         );
35.
36.         return response;
37.     }
38. }
```

Klasa iz prethodnog primjera proširuje klasu *WebServiceGatewaySupport* koja se super klasa za sve klase koje trebaju pristup SOAP Web servisima. Super klasa pruža metodu *getWebServiceTemplate()* koja vraća instancu klase *WebServiceTemplate*. Pomoću te instance omogućeno je slanje zahtjeva i primanje odgovora od SOAP Web servisa. Prethodno generirane klase služe kao objekti zahtjeva i odgovora SOAP Web servisa.

Kako bi se serijalizirali i deserijalizirali XML zahtjevi i odgovori SOAP Web servisa Spring koristi klasu *Jaxb2Marshaller*. Potrebno se postaviti *Marshaller* i *Unmarshaller* za prethodno implementiranog klijenta. Nakon što se postave te vrijednosti u konfiguraciji omogućeno je funkcionalno korištenje SOAP Web servisa kojega pruža poslužiteljska aplikacija. Primjer postavljanja nalazi se u nastavku.

```java
package com.unizg.foi.GetInformations.soap;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.oxm.jaxb.Jaxb2Marshaller;

@Configuration
public class WsConfig {

    @Bean
    public Jaxb2Marshaller marshaller() {
        Jaxb2Marshaller marshaller = new Jaxb2Marshaller();
        marshaller.setContextPath("com.unizg.foi.GetInformationsServices.soap");
        return marshaller;
    }

    @Bean
    public WeightClient articleClient(Jaxb2Marshaller marshaller) {
        WeightClient client = new WeightClient();
        client.setDefaultUri("http://get-information-
services.herokuapp.com/soap/weights.wsdl");
        client.setMarshaller(marshaller);
        client.setUnmarshaller(marshaller);
        return client;
    }

    @Bean
    public ObligationClient obligationClient(Jaxb2Marshaller marshaller) {
        ObligationClient client = new ObligationClient();
        client.setDefaultUri("http://get-information-
services.herokuapp.com/soap/obligations.wsdl");
        client.setMarshaller(marshaller);
        client.setUnmarshaller(marshaller);
        return client;
    }
}
```

9. Zaključak

CORBA, RMI te RMI-IIOP uveli su revoluciju kada je u pitanju dijeljenje podataka preko mreže. Uveli su mogućnost da aplikacije međusobno dijele objekte preko mreže te pritom više nije bilo problema ukoliko su dvije aplikacije bile napisane u različitim programskim jezicima. Na njihovom primjeru kasnije su nastali Web servisi. SOAP Web servisi koji se baziraju na dijeljenju XML poruka preko mreže bili su prethodnik danas popularnih REST Web servisa. SOAP Web servisi u današnje vrijeme koriste većinom samo za komunikaciju između poslovnih aplikacija dok se svijet većinom okrenuo REST Web servisima koji se baziraju na JSON formatu zapisa. Razlozi toge su što je pozivanje REST Web servisa jednostavnije te zbog JSON formata zapisa koji je fleksibilan i u većini slučajeva jednostavniji za čitanje i obrađivanje od strane programskih jezika. Još jedan od razloga većeg korištenja REST Web servisa je i u tome što nema standardiziranu sintaksu za razliku od SOAP Web servisa koji su standardizirani. Jedini problem koji nastaje kada su u pitanju REST Web servisi je njihova sigurnost koja se mora dodatno implementirati za razliku od SOAP Web servisa koji već imaju ugrađenu sigurnost. Problem sigurnosti se kod REST Web servisa rješava pomoću popularnih sigurnosnih metoda koje su standardizirane i pružaju sigurnije načine komunikacije i dijeljenja podataka između dvaju aplikacija. Ali usprkos slabijoj sigurnosti REST Web servisi danas se masovno koriste i gotovo da ne postoji neki veći sustav koji za svoj rad ne koristi barem jedan REST Web servis. Implementaciju Web servisa uvelike olakšavaju programski okviri. Jedan takav programski okvir je i Spring baziran na Java programskom jeziku i koji je danas široko korišten u softverskoj industriji. Zbog svih prije navedenih razloga Web servisi su sredstvo bez kojega današnja softverska industrija ne bi bila toliko uspješna. Oni su omogućili programerima da jedanput implementiraju poslovnu logiku koja se koristi u više aplikacija te da se ona samo poziva kada god je to potrebno. Kako se softverska industrija sve više i više razvija u budućnosti će zasigurno postojati još brži i sigurniji načini za dijeljenje podataka i komunikaciju aplikacija preko mreže.

Popis literature

[1] F. Bolton, „Pure CORBA“. Sams, 2002.
[2] R. M. R. PATTAMSETTI, *DISTRIBUTED COMPUTING IN JAVA 9.* S.l.: PACKT PUBLISHING LIMITED, 2017.
[3] „Java RMI over IIOP“. [Na internetu]. Dostupno na: https://www.oracle.com/technetwork/java/rmi-iiop-139743.html. [Pristupljeno: 31-srp-2018].
[4] C. Kochmer i E. Frandsen, *JSP and XML: integrating XML and web services in your JSP application*. Boston: Addison-Wesley, 2002.
[5] S. Weerawarana, Ur., *Web services platform architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and more*. Upper Saddle River, NJ: Prentice Hall PTR, 2005.
[6] E. Newcomer, *Understanding Web services: XML, WSDL, SOAP, and UDDI*. Boston: Addison-Wesley, 2002.
[7] B. Varanasi i S. Belida, *Spring REST: rest and web services development using Spring*. New York, NY: Apress; Springer, 2015.
[8] C. Shiflett, *HTTP developer's handbook*. Indianapolis, Ind: Sams, 2003.
[9] „Gentle introduction to WADL (in Java) - DZone Java“, *dzone.com*. [Na internetu]. Dostupno na: https://dzone.com/articles/gentle-introduction-wadl-java. [Pristupljeno: 08-kol-2018].
[10] J. Hartwell, *C# and XML primer*. California: Apress, 2017.
[11] D. Stokes, *MySQL and JSON: a practical programming guide*. 2018.
[12] „1. Introduction to Spring Framework“. [Na internetu]. Dostupno na: https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/overview.html#overview-web. [Pristupljeno: 29-kol-2018].

Popis slika

Popis tablica