Usporedba REST i SOAP web servisa na primjeru sustava web trgovine

Kosihajda, Karlo

Sveučilište Sjever

Završni rad br. 10/RINF/2024

Usporedba REST i SOAP web servisa na primjeru sustava web trgovine

Karlo Kosihajda, 0336057849

Đurđevac, rujan 2024. godine

Sveučilište Sjever
Sveučilišni centar Koprivnica
Trg dr. Žarka Dolinara 1, HR-48000 Koprivnica

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Računarstvo i informatika

STUDIJ: Stručni prijediplomski studij Računarstvo i informatika

PRISTUPNIK: Karlo Kosihajda | MATIČNI BROJ: 0336057849

DATUM: 26. 8. 2024. | KOLEGIJ: Implementacija web servisa i njihovo testiranje

NASLOV RADA: Usporedba REST i SOAP web servisa na primjeru sustava web trgovine

NASLOV RADA NA ENGL. JEZIKU: Comparison of REST and SOAP web services on the example of a web shop solution

MENTOR: Dražen Crčić | ZVANJE: predavač

ČLANOVI POVJERENSTVA:

1. mr.sc. Vladimir Stanisavljević, v.pred - predsjednik povjerenstva
2. doc.dr.sc. Tomislav Horvat - član povjerenstva
3. pred. Dražen Crčić, mag.ing. - mentor
4. doc.dr.sc. Domagoj Frank - zamjenski član
5.

Zadatak završnog rada

BROJ: 10/RINF/2024

OPIS:

Tehnologije web servisa su osnova za povezivanje različitih računalnih sustava i omogućuju razmjenu podataka putem interneta. Dva često korištena pristupa u tom kontekstu su REST i SOAP, svaki sa svojim jedinstvenim karakteristikama i prednostima. U sklopu završnog rada potrebno je napraviti analizu i usporedbu REST i SOAP web servisa unutar sustava web trgovine.

Potrebno je napraviti:
- općeniti opis i objašnjenje REST i SOAP web servisa
- definirati strukturu podataka web trgovine te kreirati osnovnu bazu podataka
- definirati API endpointe za sve akcije CRUD modela
- napraviti finalno programsko rješenje koje će omogućiti korištenje oba modela web servisa
- testirati konačno rješenje te napraviti usporedbu REST i SOAP web servisa na vlastitom projektu

ZADATAK URUČEN: 27. 8. 24. | POTPIS MENTORA:

SVEUČILIŠTE SJEVER

Očisti obrazac

Sveučilište
Sjever

IZJAVA O AUTORSTVU

Završni/diplomski/specijalistički rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, Karlo Kosihajda *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog/specijalističkog (*obrisati nepotrebno*) rada pod naslovom Usporedba REST i SOAP web servisa na primjeru sustava web trgovine *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

Karlo Kosihajda

(vlastoručni potpis)

Sukladno članku 58., 59. i 61. Zakona o visokom obrazovanju i znanstvenoj djelatnosti završne/diplomske/specijalističke radove sveučilišta su dužna objaviti u roku od 30 dana od dana obrane na nacionalnom repozitoriju odnosno repozitoriju visokog učilišta.

Sukladno članku 111. Zakona o autorskom pravu i srodnim pravima student se ne može protiviti da se njegov završni rad stvoren na bilo kojem studiju na visokom učilištu učini dostupnim javnosti na odgovarajućoj javnoj mrežnoj bazi sveučilišne knjižnice, knjižnice sastavnice sveučilišta, knjižnice veleučilišta ili visoke škole i/ili na javnoj mrežnoj bazi završnih radova Nacionalne i sveučilišne knjižnice, sukladno zakonu kojim se uređuje umjetnička djelatnost i visoko obrazovanje.

Sveučilište Sjever

Odjel za računarstvo i informatiku

Završni rad br. 10/RINF/2024

Usporedba REST i SOAP web servisa na primjeru sustava web trgovine

Student

Karlo Kosihajda, 0336057849

Mentor

Dražen Crčić, mag.ing.el.

Đurđevac, rujan 2024. godine

Sažetak

Ovaj rad je usredotočen na usporedbu dvaju popularnih pristupa komunikaciji između računala u sustavu web trgovine, a to su REST i SOAP. Prvo se razmatraju općenite karakteristike tih web servisa i njihova usporedba, zatim struktura podataka web trgovine (preko entiteta kao što su korisnici, proizvodi, narudžbe) te se na kraju vrši usporedba navedenih tehnologija na konkretnom razvijenom programskom rješenju. Kroz analizu programskih kodova i implementaciju REST i SOAP servisa na sličnim primjerima, detaljno se prikazuju razlike u njihovoj primjeni. U završnom dijelu uspoređuju se performanse i daju se zaključci o prednostima i nedostacima oba pristupa u kontekstu web trgovine.

Zaključak rada ukazuje na ključne razlike između REST i SOAP tehnologija. REST se ističe jednostavnošću i brzinom, što ga čini pogodnim za lagane aplikacije, dok SOAP nudi bolju sigurnost i pouzdanost, zbog čega se preferira u složenijim sustavima s osjetljivim podacima.

Ključne riječi: REST, SOAP, web servisi, e-trgovina, komunikacija, podaci.

This paper focuses on the comparison of two popular approaches to communication between computers in an e-commerce system, namely REST and SOAP. First, the general characteristics of these web services and their comparison are discussed, followed by the data structure of the e-commerce system (through entities such as users, products, orders), and finally, the comparison of the mentioned technologies is performed on a specific developed software solution. Through the analysis of code and the implementation of REST and SOAP services on similar examples, the differences in their application are presented in detail. In the final part, the performance is compared, and conclusions are drawn about the advantages and disadvantages of both approaches in the context of e-commerce.

The conclusion highlights the key differences between REST and SOAP technologies. REST stands out for its simplicity and speed, making it suitable for lightweight applications, while SOAP offers better security and reliability, making it preferred in more complex systems with sensitive data.

Keywords: REST, SOAP, web services, e-commerce, communication, data.

Popis korištenih kratica

REST	Reprezentativni prijenos stanja
SOAP	Jednostavni protokol za razmjenu objekata
API	Sučelje za programske aplikacije
CRUD	Stvaranje, Čitanje, Ažuriranje, Brisanje
URL	Uniformni resursni lokator
HTTP	Protokol za prijenos hiperteksta
HTTPS	Sigurni protokol za prijenos hiperteksta
XML	Proširivi jezični format
JSON	JavaScript notacija objekata
SMTP	Jednostavni protokol za prijenos pošte
WSDL	Jezični opis web servisa
TCP	Protokol za kontrolu prijenosa
UDP	Protokol korisničkih datagrama
JS	Javaskripta
PHP	Pretprocesor hiperteksta
ER	Entitetsko-relacijski

REST	Representational State Transfer
SOAP	Simple Object Access Protocol
API	Application Programming Interface
CRUD	Create, Read, Update, Delete
URL	Uniform Resource Locator
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
XML	Extensible Markup Language
JSON	JavaScript Object Notation
SMTP	Simple Mail Transfer Protocol
WSDL	Web Service Description Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
JS	Javascript
PHP	Hypertext Preprocessor
ER	Entity–relationship

Sadržaj

1. Uvod

U svijetu e-trgovine, učinkovita komunikacija između sustava postala je ključna komponenta za uspjeh. Web servisi pružaju temelj za povezivanje različitih računalnih sustava i omogućuju razmjenu podataka. Dva često korištena pristupa u tom kontekstu su REST (engl. *Representational State Transfer*) i SOAP (engl. *Simple Object Access Protocol*), svaki sa svojim jedinstvenim karakteristikama i prednostima. REST se temelji na HTTP protokolu i uglavnom koristi JSON format za razmjenu podataka, dok SOAP koristi XML i može raditi preko više protokola poput HTTP, TCP, pa čak i SMTP-a.

SOAP je razvijen kao robusniji protokol, dizajniran za rad u složenim i distribuiranim sustavima, s naprednim sigurnosnim i transakcijskim mogućnostima. S druge strane, REST se ističe jednostavnošću i fleksibilnošću, omogućavajući brzu razmjenu podataka i lakoću implementacije u modernim web aplikacijama [1].

U ovom radu, detaljno će se analizirati i usporediti REST i SOAP web servisi unutar sustava web trgovine. Kroz rad će se prikazati kako se implementiraju osnovne CRUD operacije koristeći oba pristupa. Fokus će biti na analizi strukture podataka web trgovine, koja uključuje entitete poput korisnika, proizvoda i narudžbi, te na implementaciji API krajnjih točki komunikacije (engl. *endpoint*) za operacije kao što su prijava, registracija, pregled i dodavanje proizvoda. Također će se prikazati kako oba pristupa upravljaju sigurnošću i autentifikacijom te kako se obrađuju zahtjevi na klijentskoj i poslužiteljskoj strani.

Poseban naglasak stavljen je na usporedbu performansi, pouzdanosti i složenosti implementacije oba pristupa. Analizirani su faktori kao što su potrebna količina koda, troškovi implementacije i održavanja, brzina izvršavanja zahtjeva te količina pogrešaka u radu s podacima. Rezultati ovog istraživanja pružit će temelj za izbor između REST i SOAP tehnologija, ovisno o potrebama specifične web aplikacije. Usporedba će se temeljiti na stvarnim primjerima korištenja unutar sustava web trgovine, uzimajući u obzir prednosti i nedostatke oba pristupa.

2. REST i SOAP

REST i SOAP arhitekture služe kao okviri za izgradnju API-ja, omogućujući različitim sustavima međusobnu interakciju putem mreže. Iako imaju zajednički cilj, razlikuju se po načinu na koji postižu ovaj cilj, koristeći različite pristupe i standarde. U nastavku će se detaljno istražiti principi i karakteristike obje arhitekture, te njihove prednosti i nedostatke.

2.1. API

API (aplikacijsko programsko sučelje, engl. *Application Programming Interface*) predstavlja skup definicija i protokola za izgradnju te integraciju programske opreme. Ponekad se naziva i dogovorem između davatelja informacija i korisnika informacija. Unutar API komunikacije se utvrđuje sadržaj koji je potreban krajnjem korisniku (poziv) i sadržaj koji je potreban proizvođaču (odgovor). Na primjer, dizajn API-ja za uslugu vremenske prognoze može odrediti da korisnik mora dostaviti poštanski broj, a proizvođač odgovara s dvodijelnim odgovorom, pri čemu je prvi dio visoka temperatura, a drugi niska [2].

Drugim riječima, ako je željna komunikacija s računalom ili sustavom radi dohvaćanja informacija ili izvršavanja funkcije, API omogućava korisnicima komunikaciju s tim sustavom kako bi ga mogao razumjeti i ispuniti zahtjev. API se može smatrati posrednikom između korisnika ili klijenata te resursa ili web usluga koje žele dobiti. Također je način za organizaciju dijeljenja resursa i informacija uz održavanje sigurnosti, kontrole i autentifikacije – određivanje tko ima pristup čemu. Još jedna prednost API-ja jest da nije potrebno znati pojedinosti o predmemoriranju, kako se resurs dohvaća ili odakle dolazi.

2.2. REST

REST (engl. *Representational State Transfer*) je softverska arhitektura koja nameće uvjete o tome kako API treba raditi. REST je prvotno stvoren kao smjernica za upravljanje komunikacijom na složenoj mreži poput interneta. Arhitektura temeljena na REST-u za podršku visokoučinkovite i pouzdane komunikacije se može koristiti na velikom broju praktičnih slučajeva. Jednostavno se implementira i modificira, donoseći vidljivost i prenosivost na više platformi svakom API sustavu.

API programeri mogu dizajnirati API komponente koristeći nekoliko različitih arhitektura. API komponente koje slijede REST arhitektonski stil nazivaju se REST API. Web usluge koje implementiraju REST arhitekturu nazivaju se RESTful web usluge. Izraz RESTful API obično se koristi za označavanje web servisa koji slijede REST arhitekturu. Međutim, pojmovi REST API i RESTful API mogu se koristiti naizmjenično [2].

Neki od osnovnih principa REST arhitekture su jedinstveno sučelje, komunikacija bez održavanja stanja, višeslojni sustav, predmemoriranje te prijenos softverskog koda klijentu. U nastavku će se ti principi detaljno opisati.

Jedinstveno sučelje (engl. *Uniform interface*) ključan je princip je za dizajn bilo kojeg RESTful web servisa. Ono ukazuje da poslužitelj prenosi informacije u standardnom formatu. Formatirani resurs naziva se reprezentacijom u REST-u. Taj format može biti različit od unutarnje reprezentacije resursa na aplikaciji poslužitelja. Na primjer, poslužitelj može pohranjivati podatke kao tekst, ali ih slati u HTML formatu reprezentacije.

Jedinstveno sučelje nameće četiri arhitektonska ograničenja navedena u nastavku.

1. Zahtjevi bi trebali identificirati resurse. To postižu korištenjem jedinstvenog identifikatora resursa.
2. Klijenti imaju dovoljno informacija u reprezentaciji resursa da ih mogu mijenjati ili brisati. Poslužitelj zadovoljava ovaj uvjet slanjem metapodataka koji dodatno opisuju resurs.
3. Klijenti dobivaju informacije o tome kako dalje obraditi reprezentaciju. Poslužitelj to postiže slanjem samoopisnih poruka koje sadrže metapodatke o tome kako klijent najbolje može koristiti te informacije.
4. Klijenti dobivaju informacije o svim ostalim povezanim resursima koje im je potrebno za dovršavanje zadatka. Poslužitelj to postiže slanjem hiperlinkova u reprezentaciji kako bi klijenti dinamički otkrivali više resursa.

U REST arhitekturi, **komunikacija bez održavanja stanja** (engl. *statelessness*) se odnosi na komunikacijsku metodu u kojoj poslužitelj dovršava svaki zahtjev klijenta neovisno o svim prethodnim zahtjevima. Klijenti mogu zatražiti resurse u bilo kojem redoslijedu i svaki zahtjev je izoliran od drugih zahtjeva. Ovo REST API ograničenje implicira da poslužitelj može u potpunosti razumjeti i ispuniti zahtjev svaki put.

U **višeslojnom sustavu** (engl. *layered system*) klijent se može povezivati s drugim ovlaštenim posrednicima između klijenta i poslužitelja, a i dalje će primati odgovore od poslužitelja. Poslužitelji također mogu proslijediti zahtjeve drugim poslužiteljima. Može se dizajnirati RESTful web servis da radi na nekoliko poslužitelja s više slojeva poput sigurnosti, aplikacije i poslovne logike koje zajedno rade kako bi ispunile zahtjeve klijenta. Ovi slojevi ostaju nevidljivi klijentu.

RESTful web servisi podržavaju **predmemoriranje** (engl. *caching*), proces pohrane određenih odgovora na klijentu ili na posredniku kako bi se poboljšalo vrijeme odgovora poslužitelja. Taj proces smanjuje ukupan broj interakcija između klijenta i poslužitelja. Koristi se najčešće kod

zaglavlja i podnožja koji se pojavljuju ili sličnih informacijskih struktura. Sam web poslužitelj mora u odgovoru implicitno ili eksplicitno prezentirati sebe sa statusom za predmemoriranje (engl. *cacheable*/ *noncacheable*).

U REST arhitektonskom stilu, poslužitelji mogu privremeno proširivati ili prilagođavati funkcionalnosti klijenta prijenosom softverskog programskog koda klijentu. Na primjer, kada se popunjava obrazac za registraciju na bilo kojoj web stranici, preglednik odmah ističe bilo kakve pogreške koje se naprave, poput pogrešnih brojeva telefona. To može učiniti zbog koda koji je poslao poslužitelj. Taj kod se naziva **izvorni kod za zahtjev** (engl. *code on demand*).

Osnovna funkcija RESTful API sučelja usporediva je s konceptom pregledavanja interneta. Kada klijentu treba određeni resurs, on kontaktira poslužitelja putem API-ja. API programeri pružaju smjernice o tome kako klijent treba integrirati s REST API-jem u dokumentaciji poslužiteljske aplikacije. Opći koraci za svaki REST API poziv su dani u nastavku.

1. Slanje zahtjeva klijenta poslužitelju - klijent konzultira API dokumentaciju kako bi oblikovao zahtjev na način koji poslužitelj može razumjeti.
2. Autentifikacija klijenta i potvrda prava - poslužitelj autentificira klijenta i provjerava ima li klijent ovlasti za upućivanje tog određenog zahtjeva.
3. Obrada zahtjeva na strani poslužitelja - poslužitelj prima zahtjev i obrađuje ga prema unutarnjim postavkama.
4. Povratni odgovor poslužitelja klijentu - poslužitelj šalje odgovor klijentu, uključujući informacije koje klijentu daju povratnu informaciju o uspješnosti zahtjeva. Osim toga, odgovor sadrži sve tražene informacije koje je klijent zatražio.

Detalji zahtjeva i odgovora za REST API mogu varirati ovisno o dizajnu API-ja koji su programeri definirali.

RESTful API sučelja postavljaju zahtjeve koji moraju sadržavati ključne komponente navedene u nastavku.

- **Jedinstveni identifikator izvora** - poslužitelj identificira svaki resurs pomoću jedinstvenih identifikatora resursa. Za REST usluge, uobičajeno je koristiti jedinstveni lokator resursa (URL) kao identifikator. URL specificira put do resursa, što je slično adresi web stranice koju unosite u preglednik za posjet određenoj web stranici. URL, također poznat kao krajnja točka zahtjeva, jasno definira poslužitelju što klijent traži.
- **Metoda** - RESTful API često koriste protokol za prijenos hiperteksta (HTTP), a metoda HTTP-a informira poslužitelj o radnji koja se treba izvršiti nad resursom. Četiri uobičajene HTTP metode su ***GET*** (pristup resursima na navedenom URL-u), ***POST***

(slanje podataka na poslužitelj), ***PUT*** (ažuriranje postojećih resursa) i ***DELETE*** (uklanjanje resursa).

- **HTTP zaglavlja** - zaglavlja zahtjeva prenose metapodatke između klijenta i poslužitelja. Na primjer, zaglavlje zahtjeva može označavati format zahtjeva i odgovora, pružiti informacije o statusu zahtjeva itd.
- **Podaci** - REST API zahtjevi mogu uključivati podatke potrebne za POST, PUT i druge HTTP metode kako bi se uspješno izvršile.
- **Parametri** - RESTful API zahtjevi mogu sadržavati različite vrste parametara kao što su parametri puta koji određuju detalje URL-a, parametri upita koji pružaju dodatne informacije o resursu i parametri kolačića koji omogućuju brzu autentifikaciju klijenata.

RESTful web usluge mogu verificirati autentičnost zahtjeva prije nego što pošalju odgovor. Autentifikacija predstavlja proces provjere identiteta, analogno kao što se može dokazati identitet pokazivanjem osobne iskaznice ili vozačke dozvole. Slično tome, klijenti RESTful usluge moraju dokazati svoj identitet poslužitelju kako bi uspostavili povjerenje.

RESTful API koristi četiri uobičajene metode provjere autentičnosti koje su navedene u nastavku.

1. HTTP provjera autentičnosti - HTTP definira nekoliko shema provjere autentičnosti koje se mogu izravno koristiti prilikom implementacije REST API-ja. Primjerice, osnovna autentifikacija uključuje slanje korisničkog imena i lozinke u zaglavlju zahtjeva, kodirane base64 tehnikom za siguran prijenos.
2. API ključevi - API ključevi su još jedna opcija za autentifikaciju u REST API-ju. Poslužitelj dodjeljuje jedinstveni API ključ prvom klijentu, a klijent ga koristi za provjeru prilikom pristupa resursima. Unatoč svojoj upotrebljivosti, API ključevi su manje sigurni jer se moraju prenositi, čime se otvara mogućnost mrežne krađe.
3. Oauth - OAuth kombinira lozinke i tokene kako bi osigurao vrlo siguran pristup sustavu. Poslužitelj prvo traži lozinku, a zatim dodatni token za dovršetak procesa autorizacije. Token se može provjeriti u bilo kojem trenutku, s određenim opsegom i dugotrajnošću.
4. Autentifikacija nositelja - Autentifikacija nositelja odnosi se na davanje kontrole pristupa nositelju tokena. Token nositelja, generiran od strane poslužitelja kao odgovor na zahtjev za prijavu, šalje se u zaglavlju zahtjeva za pristup resursima. Ovaj pristup pomaže u osiguravanju sigurnosti i pouzdanosti autentifikacije.

Principi REST-a zahtijevaju da odgovor poslužitelja sadrži tri ključne komponente. Te ključne komponente su statusna linija, tijelo poruke i zaglavlja.

Statusna linija sadrži troznamenkasti statusni kod koji obavještava o uspjehu ili neuspjehu zahtjeva. Primjeri statusnih kodova uključuju kodove 2XX za uspjeh, kodove 4XX i 5XX za pogreške te 3XX za preusmjeravanje URL-a. Neki od uobičajenih statusnih kodova su 200 za generički odgovor na uspjeh, zatim 201 koji predstavlja uspješan odgovor na POST metodu, statusni kod 400 koji označava netočan zahtjev koji poslužitelj ne može obraditi te statusni kod 404 koji predstavlja da resurs nije pronađen [2].

Tijelo poruke unutar odgovora sadrži prikaz resursa, a poslužitelj odabire format reprezentacije na temelju sadržaja zaglavlja zahtjeva. Klijenti mogu zatražiti informacije u XML ili JSON formatima koji definiraju kako se podaci predstavljaju u običnom tekstu [2].

Odgovor također sadrži zaglavlja ili metapodatke o odgovoru. Zaglavlja pružaju dodatne informacije o odgovoru, uključujući podatke o poslužitelju, kodiranju, datumu i vrsti sadržaja [2].

Kako bi se ilustrirao rad REST arhitekture, može se uzeti primjer dohvaćanja određenog resursa korištenjem HTTP metode GET. U ovom slučaju, želi se dohvatiti film čiji je ID jednak 3 iz resursa */movies*. Kada klijent pošalje zahtjev prema poslužitelju na URL */movies/3*, poslužitelj odgovara slanjem JSON reprezentacije traženog filma. Odgovor, koji sadrži relevantne podatke o filmu, kao što su ID i naziv bi izgledao kao što je prikazano u nastavku.

```
{"id": 3, "title": "Sam u kući 2"}
```

2.2.1. Prednosti

Neke od prednosti RESTful web servisa u odnosu na ostale tehnologije su jednostavnost, prilagođenost mrežnoj arhitekturi, mogućnost predmemoriranja, skalabilnost, učinkovitost, neovisnost o korištenoj tehnologiji i fleksibilnost.

Jednostavnost – REST je dizajniran za rad na klijentima koji imaju ograničeni resursi, od web preglednika, različitih vrsta mrežne opreme, na mnoštvo različitih vrsta usluga. Zbog zahtjeva za jednostavnim i fleksibilna implementacija je vrlo jednostavna i slična velikoj većini mrežni promet.

Pogodnost infrastrukture - vatrozidi i poslužitelji optimizirani su za REST arhitekturu jer su optimizirani za HTTP/HTTPS promet.

Mogućnost predmemoriranja - neke vrste REST zahtjeva su prihvatljive za keširanje. Mrežna infrastruktura namijenjena predmemoriranju može odgovoriti točnim odgovorom.

Skalabilnost - sustavi koji implementiraju REST API-je mogu se učinkovito skalirati jer REST optimizira interakcije klijent-poslužitelj. Komunikacija bez održavanja stanja uklanja opterećenje

poslužitelja jer poslužitelj ne mora zadržati podatke o prethodnim zahtjevima klijenta. Dobro upravljano predmemoriranje djelomično ili potpuno eliminira neke interakcije klijent-poslužitelj. Sve te značajke podržavaju skalabilnost bez smanjenja performanse..

Učinkovitost - dok SOAP web usluge uvijek vraćaju XML dokumente, REST web usluge omogućuju fleksibilnost u smislu vrste podataka koje vraćaju. Zadani format za prijenos korisnih podataka sadržaji su JSON objekti. Prve verzije mobilne platforme Android nisu uključujući okvire za raščlanjivanje JSON objekata.

Neovisnosti - REST API-ji neovisni su o korištenoj tehnologiji. Mogu se pisati i klijentske i poslužiteljske aplikacije na različitim programskim jezicima bez utjecaja na dizajn API-ja. Također mogu se promijeniti temeljne tehnologije na bilo kojoj strani bez utjecaja na komunikaciju.

Fleksibilnost - RESTful web usluge podržavaju potpuno odvajanje klijent-poslužitelj. Oni pojednostavljuju i odvajaju različite komponente poslužitelja tako da se svaki dio može samostalno razvijati. Promjene platforme ili tehnologije na poslužiteljskoj aplikaciji ne utječu na klijentsku aplikaciju. Mogućnost slojevitosti aplikacijskih funkcija dodatno povećava fleksibilnost. Na primjer, programeri mogu mijenjati sloj baze podataka bez ponovnog pisanja logike aplikacije.

2.2.2. Nedostaci

Iako sadrži niz prednosti u odnosu na slične tehnologije, REST ima i određene nedostatke koji se najviše očituju u sigurnosnom aspektu koji nije predefiniran, nego je o njemu potrebno voditi računa tijekom razvoja i implamentacije, a uz to REST nema strogo definiran standard

Sigurnost mora biti uključena u proces planiranja – kao i kod svakog drugog podatkovnog protokola, sigurnost također mora biti dobro planirana za REST arhitekturu. Potrebno je imati na umu mobilne podatke aplikacija, jer se povećava i broj mobilnih aplikacija mogućnost napada i zlostavljanja. Potrebno je pažljivo ispitati koji podaci prenosit će se između klijenta i web usluge. Informacije o identitet korisnika ne smije se prenositi na mobilni uređaj, osim Ako je to prijeko potrebno.

Nije strogo definiran standard - postoje mnoge implementacije, označene kao REST web usluge, koje zapravo nisu ništa drugo nego obični servlet koji kontrolira objavljeni sadržaj i izvodi njegove operacije. Ovo je pogrešan pristup koji ne iskorištava mnoge prednosti arhitekture. Uključivanje parametara naredbe u URL zahtjev za omogućavanje praćenje sustava i upravljanje zahtjevima. Korištenje cijelog seta HTTP metode (POST, GET, PUT, DELETE) omogućuju funkcionalnost razdvajanje online usluge na različite dijelove.

2.3. SOAP

SOAP (engl. *Simple Object Access Protocol*), predstavlja protokol za web usluge koji koristi XML za olakšavanje prijenosa podataka i dokumenata putem HTTP-a ili SMTP-a (engl. *Simple Mail Transfer Protocol*). Ovaj protokol je osmišljen za Microsoft još 1998. godine, a danas se široko primjenjuje u izlaganju web usluga i transferu podataka putem HTTP/HTTPS [8].

Jedna od ključnih značajki ugrađene funkcionalnosti u SOAP-u jest stvaranje web-baziranih usluga. Ova funkcionalnost omogućuje SOAP-u da učinkovito upravlja komunikacijama, pri čemu odgovori postaju neovisni o jeziku i platformi. Ova karakteristika čini SOAP svestranim i prilagodljivim za širok raspon implementacija, čime olakšava interoperabilnost u heterogenim okruženjima. SOAP poruke se sastoje od četiri ključna bloka, a to su omotnica, zaglavlje, tijelo i pogreška.

Omotnica (engl. *envelope*) je osnovna komponenta svake SOAP poruke. Poruka počinje i završava svojim oznakama, obuhvaćajući ih, što objašnjava naziv "omotnica" (engl. *envelope*). Cijela SOAP poruka smješta se unutar ove omotnice, uključujući preostala tri bloka informacija.

Zaglavlje (engl. *header*) je opcionalni element koji određuje specifičnosti i dodatne zahtjeve poruke, primjerice, ovjera. Iako nije obavezno, "soap:Header" omogućuje proširenje SOAP-a putem SOAP modula. Ovi moduli mogu biti obavezni ili opcionalni.

Tijelo (engl. *body*) sadrži zahtjev ili odgovor. Prostori imena mogu se koristiti za opisivanje podataka koje možete očekivati unutar tijela, iako nisu obavezni. U praksi, naziv procedure, parametri i podaci prolaze kroz tijelo SOAP poruke.

Pogreška (engl. *fault*) je također opcionalni element koji prikazuje sve informacije o mogućim pogreškama tijekom API zahtjeva i odgovora. Različiti uzroci pogrešaka mogu uključivati netočno SOAP formatiranje, greške u obradi na poslužitelju ili neslaganje vrsta podataka.

```
<soap:Envelope   xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:web="http://www.example.com/webservice">
   <soap:Header>
   </soap:Header>
   <soap:Body>
      <web:GetUserInfo>
         <web:UserID>123</web:UserID>
      </web:GetUserInfo>
   </soap:Body>
</soap:Envelope>
```

Na prethodnom izvornom kodu je prikazan jednostavan primjer SOAP-a. U primjeru se vidi kako *Envelope* označava početak i kraj SOAP poruke, dok se u *Body* izvršava stvarni zahtjev, u ovom slučaju se traže informacije od korisnika koristeći njihov ID.

2.3.1. WSDL

WSDL (engl. *Web Service Description Language*) je jezik koji definira ponašanje i radnje web servisa u skladu sa SOAP porukama, kao i obrnuto. Ovaj jezik postavlja apstraktne smjernice prema kojima svaki web servis obrađuje dolazne poruke i generira odgovore. Dodatno, WSDL opisuje sučelja kojima korisnik upravlja i djeluje nad podacima [8].

Da bi se bolje razumjela uloga WSDL jezika, važno je razumjeti koncept web servisa. Web servisi su usluge koje računala pružaju korisnicima omogućavajući im komunikaciju s drugim računalima u internetskoj mreži na objektno orijentiran način te upravljanje podacima u bazi podataka putem definiranog sučelja. Drugim riječima, web servisi omogućuju prikaz sučelja koja korisnicima služe za interakciju s aplikacijama, bazama podataka i drugim korisnicima. Ove interakcije kontrolirane su pravilima zapisanim u WSDL datoteci. Neka od ovih pravila reguliraju način slanja zahtjeva web servisu, očekivane parametre koje web servis traži, te način obavljanja operacija. WSDL tako igra ključnu ulogu u definiranju standardizirane komunikacije između web servisa i korisnika.

Sama struktura WSDL dokumenta bi izgledala ovako:

```
<definitions>

<types>
..........
</types>

<message>
..........
</message>

<portType>
..........
</portType>

<binding>
..........
</binding>

<service>
..........
</service>

</definitions>
```

Jezik WSDL dijeli mnoge sličnosti s HTML jezikom u pogledu svojih oznaka, čineći ga vrlo čitljivim i jednostavnim za pisanje.

- <types> element opisuje tipove podataka koji se koriste u komunikaciji između klijenta i poslužitelja

- <messages> element opisuje podatke koji se izmjenjuju u komunikaciji klijent-poslužitelj
- <portType> element kombinira više message elemenata kako bi se kreirala jednosmjerna ili višesmjerna komunikacija
- <binding> element opisuje detalje kako će se portType operacija ustvari slati preko mreže
- <service> element definira konačne portove web servisa

2.3.2. Prednosti

SOAP predstavlja dobro utemeljenu tehnologiju koja uspješno poslužuje u razvoju servisno orijentirane arhitekture. Mnoge tvrtke su, zahvaljujući SOAP-u, uspjele uspostaviti različite razine usluga koje omogućuju aplikacijama pristup uslugama unutar i izvan vatrozida. SOAP je koncipiran kako bi iskoristio različite metode prijenosa podataka, uključujući sinkroni HTTP/HTTPS, asinkrone vrste, te čak i putem e-mailova. Zanimljivo je napomenuti da je SOAP razvijen prije pojave mobilnih uređaja i tableta.

2.3.3. Nedostaci

U nastavku su navedeni neki od nedostataka SOAP web servisa.

Kontrola promjena - Promjene u web uslugama temeljenim na SOAP-u često zahtijevaju prilagodbe na strani klijenta. Dok za web aplikacije ovo može biti manje problematično, mobilne aplikacije često susreću izazove u ažuriranju.

Složenost - Izgradnja SOAP klijenta na temelju WSDL dokumenta može biti izuzetno komplicirana. Dodatni problem nastaje kada tvrtke i programeri trebaju podržati istu aplikaciju na različitim mobilnim platformama. Ponovno prilagođavanje složenih SOAP sučelja oduzima puno vremena, često je zahtjevno i povećava vjerojatnost grešaka u programskom kodu.

Mogućnost ponovne upotrebe - Moderne web aplikacije sve više se razvijaju uz potporu REST arhitekture, što pruža mnogo mogućnosti za ponovnu upotrebu. U suprotnosti s tim, korištenje SOAP protokola za razvoj mobilnih aplikacija može ograničiti opcije ponovne upotrebe. S prelaskom mobilnih aplikacija na HTML5, važnost ponovne upotrebe sve više raste [4].

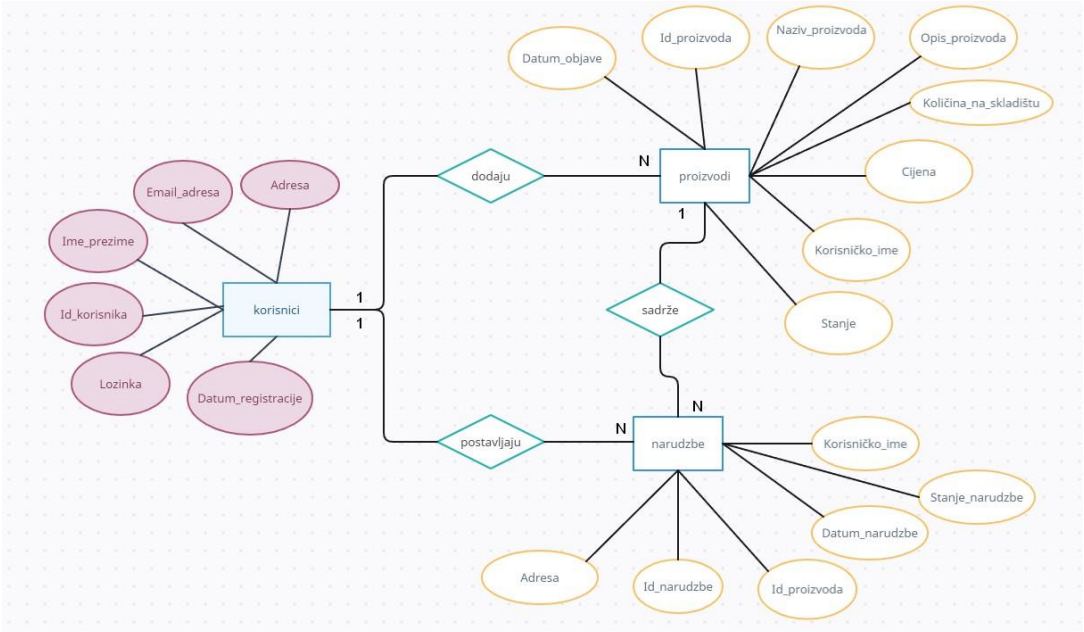
3. Struktura podataka web trgovine

Za potrebe ovog rada kreirana je potpuno nova web trgovina sa vlastitom arhitekturom i strukturom podataka. Sama struktura podataka web trgovine sastoji se od SQL baze podataka koja sadrži funkcionalno minimalni set informacija unutar tablica korisnici, proizvodi i narudžbe. Ova struktura podataka pruža osnovu za upravljanje informacijama u sustavu web trgovine. Svaka tablica sadrži ključne elemente koji omogućuju identifikaciju, praćenje i upravljanje podacima. Osim toga, navedena struktura omogućava izgradnju relacija između tablica (npr. narudžba povezana s korisnikom ili proizvodom). Takva struktura podataka podržava osnovne funkcionalnosti web trgovine poput registracije korisnika, pregleda i kupovine proizvoda te upravljanje narudžbi. Obzirom da je tema rada usporedba REST i SOAP tehnologija, zbog jednostavnosti je definirana baza u kojoj jedna narudžba može sadržavati samo jedan proizvod.

Konceptualni model predstavlja apstraktni pregled baze podataka, naglašavajući entitete i njihove odnose bez detaljnog opisa implementacije. U ovom modelu, fokus je na glavnim entitetima (korisnici, proizvodi, narudžbe) i njihovim vezama.

Logički model detaljnije opisuje strukturu podataka, uključujući atribute svakog entiteta i specifične veze između njih. Ovaj model uzima u obzir detalje kao što su ključni atributi i referencijalne veze.

U procesu dizajniranja baze podataka, konceptualni model je bio prvi korak, omogućujući identifikaciju osnovnih entiteta i njihovih odnosa. Nakon toga, logički model je razvijen kako bi se detaljno opisala struktura podataka i veze između entiteta, osiguravajući da svi zahtjevi za funkcionalnost i integritet podataka budu zadovoljeni.



Slika 3-1 ER dijagram web trgovine

Na slici je prikazan ER dijagram koji prikazuje entitete, njihove atribute i odnose između njih. Za vezu *korisnici – proizvodi* se može reći da jedan korisnik može dodati više proizvoda, ali svaki proizvod je dodao samo jedan korisnik. Za vezu *korisnici – narudzbe* se može reći da jedan korisnik može napraviti više narudžbi, ali svaka narudžba pripada samo jednom korisniku. Za vezu *proizvodi – narudzbe* se može reći da jedan proizvod može biti u više narudžbi, ali svaka narudžba sadrži jedan proizvod, zbog jednostavnosti kao što je navedeno na početku poglavlja.

3.1. Korisnici

Tablica *korisnici* predstavlja podatke o krajnjim korisnicima web trgovine, te sadržava sljedeće kolone:

- ID korisnika - jedinstveni identifikator korisnika
- Ime i prezime - osobno ime korisnika
- Korisničko ime - unikatno korisničko ime za prijavu
- Email adresa - kontakt informacija korisnika
- Adresa - fizička adresa ili adresa dostave
- Lozinka - sigurnosni ključ za pristup korisničkom računu
- Datum registracije - datum kada je korisnik registriran

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	ID_korisnika	int(11)			No	None		AUTO_INCREMENT	Change Drop More
☐	2	Ime_prezime	varchar(100)	utf8mb4_general_ci		Yes	NULL			Change Drop More
☐	3	Korisnicko_ime	varchar(50)	utf8mb4_general_ci		Yes	NULL			Change Drop More
☐	4	Email_adresa	varchar(100)	utf8mb4_general_ci		Yes	NULL			Change Drop More
☐	5	Adresa	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
☐	6	Lozinka	varchar(100)	utf8mb4_general_ci		Yes	NULL			Change Drop More
☐	7	Datum_registracije	date			Yes	NULL			Change Drop More

Slika 3.1-1 Struktura tablice korisnici

3.2. Proizvodi

Za potrebe spremanja i evidencija podataka koji se prodaju putem web trgovine koristi se tablica *proizvodi* koji se sastoji od sljedećih kolona:

- ID proizvoda - jedinstveni identifikator proizvoda
- Naziv proizvoda - ime proizvoda
- Opis proizvoda - detaljan opis proizvoda
- Cijena - cijena proizvoda
- Količina na skladištu - količina dostupnih proizvoda na skladištu

- Korisničko ime - korisničko ime od korisnika koji je dodao proizvod
- Datum objave - datum kada je proizvod objavljen
- Stanje - stanje proizvoda, da li je novo ili rabljeno

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	ID_proizvoda	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	Naziv_proizvoda	varchar(100)	utf8mb4_general_ci		Yes	NULL			Change Drop More
3	Opis_proizvoda	text	utf8mb4_general_ci		Yes	NULL			Change Drop More
4	Cijena	decimal(10,2)			Yes	NULL			Change Drop More
5	Kolicina_na_skladistu	int(11)			Yes	NULL			Change Drop More
6	Korisnicko_ime	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
7	Datum_objave	date			Yes	NULL			Change Drop More
8	Stanje	varchar(20)	utf8mb4_general_ci		Yes	NULL			Change Drop More

Slika 3.2-1 Struktura tablice proizvodi

3.3. Narudžbe

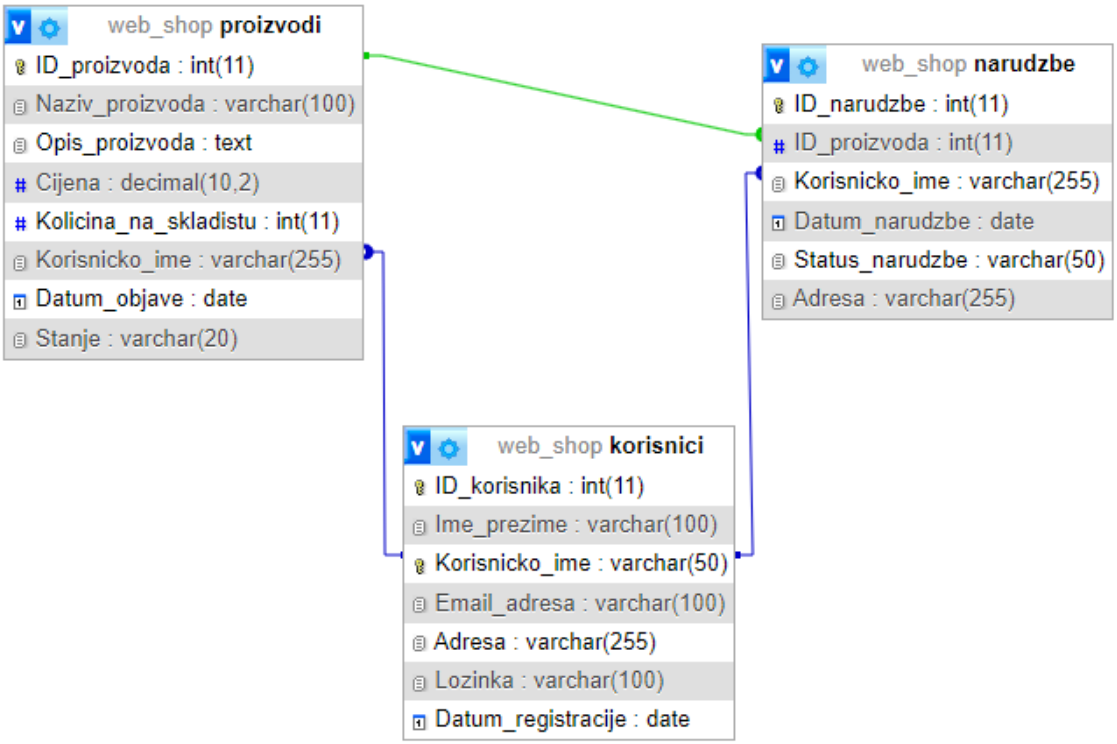
Podaci o ostvarenim narudžbama unutar web trgovine su povezani s tablicom *narudzbe* koji sadržava sljedeće kolone:

- ID narudžbe - jedinstveni identifikator narudžbe
- ID proizvoda - povezuje narudžbu s određenim proizvodom
- Korisničko ime - korisničko ime od korisnika koji je dodao proizvod pod narudžbe
- Datum narudžbe - datum kada je narudžba napravljena
- Status narudžbe - označava trenutni status narudžbe
- Adresa dostave - adresa na koju će narudžba biti dostavljena

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	ID_narudzbe	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	ID_proizvoda	int(11)			Yes	NULL			Change Drop More
3	Korisnicko_ime	varchar(255)	utf8mb4_general_ci		Yes	NULL			Change Drop More
4	Datum_narudzbe	date			Yes	NULL			Change Drop More
5	Status_narudzbe	varchar(50)	utf8mb4_general_ci		Yes	NULL			Change Drop More
6	Adresa	varchar(255)	utf8mb4_general_ci		No				Change Drop More

Slika 3.3-1 Struktura tablice narudzbe

3.4. Relacija između tablica



Slika 3.4-1 Relacije između tablica

Na slici 3.4-1 su prikazane tri glavne tablice (korisnici, proizvodi, narudžbe) i relacije između njih. U tablici *proizvodi* pri kreiranju novog zapisa (novog proizvoda), kolona *korisnicko_ime* označava korisnika koji je kreirao proizvod i kao strani ključ povezuje na tablicu *korisnici*. U tablici *narudzbe* kolona *koristnicko_ime* također označava poveznicu na tablicu korisnici dok kolona *ID_proizvoda* se povezuje na tablicu *proizvodi* i označava koji proizvod je dodan u narudžbu.

4. Primjena RESTful web servisa na web trgovinu

U ovom dijelu rada je opisano kako je implementirana mogućnost prijave i registracije korisnika i ažuriranje njihovih podataka koristeći RESTful web servise. JavaScript kodovi (*login.js, edit_account.js*) se koriste za rukovanje događajima na klijentskoj strani, dok *userscontroller.php* i *index.php* služe za obradu zahtjeva na poslužiteljskoj strani.

Pošto je glavna tema rada usporedba REST i SOAP tehnologija koje se implementiraju na poslužiteljskoj strani, na klijentskoj strani programskog rješenja se za potrebe demonstracija koriste jednostavne metode bez upotrebe posebnih sigurnosnih tehnika te također u nekim slučajevima integrirani podaci (bez korištenja zasebnih konfiguracijskih datoteka ili sl.). Produkcijski sustav web trgovine bi svakako trebao uzeti u obzir sve sigurnosne mjere zaštite osjetljivih i osobnih podataka, kako na klijentskoj tako i na poslužiteljskoj strani.

4.1. Funkcionalnosti prijave korisnika

U razvijenom programskom rješenju unutar datoteke *login.js* su razvijene funkcionalnosti za rukovanje korisničkim akcijama poput prijave i registracije. Ovaj kod koristi API za dohvat (engl. *fetch)* za slanje zahtjeva poslužitelju i upravljanje odgovorima.

Kada korisnik podnese formu za prijavu, podaci se šalju poslužitelju za provjeru. Ako su uneseni podaci ispravni, korisnik se preusmjerava na početnu stranicu aplikacije. Primjer isječka koda koji obavlja taj dio izgleda kao u nastavku.

```
document.getElementById('loginForm').addEventListener('submit',
function (event) {
    event.preventDefault();

    const username = document.getElementById('loginUsername').value;
    const password = document.getElementById('loginPassword').value;

    fetch('http://localhost:5500/users')
            .then(response => response.json())
            .then(users => {
                const user = users.find(user => user.Korisnicko_ime
=== username && user.Lozinka === password);
                if (user) {
                    alert('Uspješno prijavljeni!');
                    window.location.href = "home.html";
                    localStorage.setItem('username',
user.Korisnicko_ime);
                } else {
                    document.getElementById('loginError').textContent
= 'Netočno korisničko ime ili lozinka';
                }
            })
            .catch(error => {
                console.error('Error fetching users:', error);
```

```
                document.getElementById('loginError').textContent =
'An error occurred while fetching users';
            });
});
```

Kada korisnik podnese formu za registraciju, prikupljeni podaci se šalju poslužitelju pomoću POST zahtjeva. Poslužitelj obrađuje zahtjev i pohranjuje novog korisnika u bazu podataka. Kod za registraciju je prikazan u nastavku.

```
document.getElementById('registrationForm').addEventListener('submit'
, function (event) {
    event.preventDefault();

    const name = document.getElementById('registrationName').value;
    const username =
document.getElementById('registrationUsername').value;
    const password =
document.getElementById('registrationPassword').value;

    fetch('http://localhost:5500/users', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            "Ime_prezime": name,
            "Korisnicko_ime": username,
            "Lozinka": password
        })
    })
            .then(response => response.json())
            .then(data => {
                if (data.error) {

document.getElementById('registrationError').textContent = 'Greške
prilikom kreiranja računa';
                } else {
                    alert('Uspješno stvoren račun');

                }
            })
            .catch(error => {
                console.error('Error registering user:', error);

document.getElementById('registrationError').textContent = 'An error
occurred while registering user'; });});});
```

4.2. Funkcionalnosti ažuriranja korisnika

Funkcionalnosti potrebne za ažuriranje korisničkih profila su u programskom rješenju razvijene unutar datoteke *edit_account.js.* Navedene funkcionalnosti omogućuje korisnicima da upravljaju sa svojim podacima. Kada korisnik učita stranicu, skripta dohvaća trenutne korisničke podatke i popunjava formu. Nakon što korisnik podnese ažurirane podatke, skripta šalje PUT zahtjev poslužitelju za ažuriranje korisničkog računa.

Nakon učitavanja stranice, dohvaća se korisničko ime iz tzv. *localStorage* što je mehanizam u pregledniku koji omogućuje pohranu podataka na strani klijenta. *localStorage* omogućava trajnu pohranu podataka koji ostaju dostupni i nakon što se preglednik zatvori ili osvježi. Zatim se dohvaćaju i prikazuju trenutni korisnički podaci. Dio koda koji ažurira podatke naveden je u nastavku.

```
document.getElementById('editAccountForm').addEventListener('submit',
function (event) {
        event.preventDefault();

        const name = document.getElementById('editName').value;
        const email = document.getElementById('editEmail').value;
        const address = document.getElementById('editAddress').value;
        const password =
document.getElementById('editPassword').value;

        // Koristi korisničko ime koje smo ranije dobili iz lokalne
pohrane
        fetch(`http://localhost:5500/users/${username}`, {
            method: 'PUT',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                "Ime_prezime": name || null,
                "Email_adresa": email || null,
                "Adresa": address || null,
                "Lozinka": password || null
            })
        })
                .then(response => response.json())
                .then(data => {
                    if (data.error) {

document.getElementById('editAccountError').textContent = data.error;
                    } else {
                        alert('Promijene uspješno spremljene!');
                        // Redirect or perform other actions upon
successful account update
                    }
                })
                .catch(error => {
                    console.error('Error updating account:', error);
```

```
document.getElementById('editAccountError').textContent = 'An error
occurred while updating account';});});
```

Ovaj kod uzima vrijednosti iz polja za unos (editName...itd) i šalje PUT zahtjev koji onda sprema te vrijednosti. Na poslužiteljskoj strani se koristi klasa koja rukuje svim zahtjevima povezanim s korisnicima, a spremljena je u datoteku *userscontroller.php*. Koristi se za obradu HTTP zahtjeva za dohvaćanje, kreiranje i ažuriranje korisnika.

Dio koda naveden u nastavku predstavlja metodu koja dohvaća sve korisnike iz baze podataka i vraća ih u JSON formatu.

```
public function getUsers() {
        $sql = "SELECT * FROM korisnici";
        $result = $this->conn->query($sql);
        $users = [];

        while ($row = $result->fetch_assoc()) {
            $users[] = $row;
        }

        return $users;
    }
```

Kod u nastavku predstavlja metodu koja prima podatke o novom korisniku, provjerava da li već postoji korisnik s istim korisničkim imenom i, ako ne postoji, dodaje ga u bazu podataka.

```
public function createUser($data) {
        $ime_prezime = isset($data['Ime_prezime']) ?
$data['Ime_prezime'] : null;
        $korisnicko_ime = isset($data['Korisnicko_ime']) ?
$data['Korisnicko_ime'] : null;
        $email_adresa = isset($data['Email_adresa']) ?
$data['Email_adresa'] : null;
        $adresa = isset($data['Adresa']) ? $data['Adresa'] : null;
        $lozinka = isset($data['Lozinka']) ? $data['Lozinka'] : null;

        // Priprema upita s pripremljenim izjavama
        $query = "INSERT INTO korisnici (Ime_prezime, Korisnicko_ime,
Email_adresa, Adresa, Lozinka, Datum_registracije)
              VALUES (?, ?, ?, ?, ?, NOW())";

        // Priprema izjave
        $statement = mysqli_prepare($this->conn, $query);

        // Povezivanje parametara s izjavom
        mysqli_stmt_bind_param($statement, "sssss", $ime_prezime,
$korisnicko_ime, $email_adresa, $adresa, $lozinka);

        // Izvršavanje izjave
        if (mysqli_stmt_execute($statement)) {
            return ["message" => "User created successfully"];
        } else {
```

```
            return ["error" => "Error creating user: " .
mysqli_error($this->conn)];
        }
    }
```

Sljedeći kod predstavlja metodu koja ažurira podatke korisnika u bazi na temelju korisničkog imena.

```
    public function updateUser($username, $data) {
        $ime_prezime = mysqli_real_escape_string($this->conn,
$data['Ime_prezime']);
        $email_adresa = mysqli_real_escape_string($this->conn,
$data['Email_adresa']);
        $adresa = mysqli_real_escape_string($this->conn,
$data['Adresa']);
        $lozinka = mysqli_real_escape_string($this->conn,
$data['Lozinka']);

        $query = "UPDATE korisnici
              SET Ime_prezime='$ime_prezime',
Email_adresa='$email_adresa',
              Adresa='$adresa', Lozinka='$lozinka'
              WHERE Korisnicko_ime='$username'";

        if (mysqli_query($this->conn, $query)) {
            return ["message" => "User updated successfully"];
        } else {
            return ["error" => "Error updating user: " .
mysqli_error($this->conn)];
        }
    }
```

4.3. Rukovanje zahtjevima

Datoteka *index.php* služi kao ulazna točka za sve HTTP zahtjeve prema poslužitelju. Ova skripta koristi Leaf PHP razvojno okruženje za usmjeravanje zahtjeva na odgovarajuće metode u *UsersController* klasi. Kod u nastavku izvršava rukovanje zahtjevima.

```
// spajanje na bazu
$conn = new mysqli("localhost", "root", "", "web_shop");
if ($conn->connect_error) {
    die("DB connection failed: " . $conn->connect_error);
}
$usersController = new App\Controllers\UsersController($conn);
$ordersController = new App\Controllers\OrdersController($conn);
$productsController = new App\Controllers\ProductsController($conn);
// korisnici ----------------------------------------------------------
app()->get('/users', function () use ($usersController) {
    response()->json($usersController->getUsers());
});
app()->get('/users/{username}/products', function ($username) use
($productsController) {
```

```
    response()->json($productsController>getUserProducts($username));
});
app()->get('/users/{username}', function ($username) use
($usersController) {
    $user = $usersController->getUserByUsername($username);
    response()->json($user);
});
app()->post('/users', function () use ($usersController) {
    $data = json_decode(file_get_contents("php://input"), true);
    response()->json($usersController->createUser($data));
});
app()->put('/users/{id}', function ($id) use ($usersController) {
    $data = json_decode(file_get_contents("php://input"), true);
    response()->json($usersController->updateUser($id, $data));
});
app()->put('/user/{username}/address', function ($username) use
($usersController) {
    $data = json_decode(file_get_contents("php://input"), true);
    response()->json($usersController->updateUserAddress($username,
$data));
});
app()->delete('/users/{id}', function ($id) use ($usersController) {
    response()->json($usersController->deleteUser($id));
});
```

U kodu se mogu vidjeti sve pristupne točke (engl. *endpoint*) za korisnike. U nastavku su oni navedeni.

- GET /users - Dohvaća popis svih korisnika.
- GET /users{username}/products - Dohvaća popis proizvoda od određenog korisnika prema korisničkom imenu.
- GET /users{username} - Dohvaća korisnika po njegovom korisničkom imenu
- POST /users - Stvara novog korisnika.
- PUT /users/{id} - Ažurira informacije o određenom korisniku prema ID-u
- PUT /user{username}/address - Ažurira samo adresu od traženog korisnika prema korisničkom imenu.
- DELETE /users/{id} - Briše određenog korisnika prema ID-u.

5. Primjena SOAP web servisa na web trgovinu

U ovom dijelu će se opisati kako je implementirana mogućnost pregleda, dodavanja i ažuriranja proizvoda koristeći SOAP web servise. JavaScript kodovi (*add_product_soap.js*, *edit_products_soap.js*) se koriste za rukovanje događajima na klijentskoj strani, dok *soap_server.php* služi za obradu zahtjeva na poslužiteljskoj strani.

5.1. Funkcionalnosti dodavanja proizvoda

Programski kod unutar *add_product_soap.js* je odgovoran za dodavanje novih proizvoda. On prikuplja podatke iz obrasca, stvara SOAP zahtjev i šalje ga na poslužitelj.

Kod koji izvršava tu funkcionalnost je prikazan u nastavku.

```
document.addEventListener('DOMContentLoaded', function () {
    const username = localStorage.getItem('username');

document.getElementById('createProductForm').addEventListener('submit
', function (event) {
        event.preventDefault();

        const naziv_proizvoda =
document.getElementById('naziv_proizvoda').value;
        const opis_proizvoda =
document.getElementById('opis_proizvoda').value;
        const cijena = document.getElementById('cijena').value;
        const kolicina_na_skladistu =
document.getElementById('kolicina_na_skladistu').value;
        const stanje = document.getElementById('stanje').value;

        const soapRequest = `
            <soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
                <soap:Body>
                    <createProduct xmlns="http://localhost:5500">

<naziv_proizvoda>${naziv_proizvoda}</naziv_proizvoda>

<opis_proizvoda>${opis_proizvoda}</opis_proizvoda>
                        <cijena>${cijena}</cijena>

<kolicina_na_skladistu>${kolicina_na_skladistu}</kolicina_na_skladist
u>
                        <korisnicko_ime>${username}</korisnicko_ime>
                        <stanje>${stanje}</stanje>
                    </createProduct>
                </soap:Body>
            </soap:Envelope>`;

        fetch('http://localhost:5500/public/soap_server.php', {
            method: 'POST',
            headers: {
                'Content-Type': 'text/xml; charset=utf-8',
```

```
                'SOAPAction': 'http://localhost:5500#createProduct'
            },
            body: soapRequest
        })
                .then(response => response.text())
                .then(str => new
window.DOMParser().parseFromString(str, "text/xml"))
                .then(data => {
                    const response =
data.getElementsByTagName("return")[0].textContent;
                    if (response.includes("Error")) {
                        alert('Error creating product: ' + response);
                    } else {
                        alert('Proizvod uspješno dodan!');
                        window.location.href = "home.html";
                    }
                })
                .catch(error => {
                    console.error('Error creating product:', error);
                    alert('Greška pri dodavanju proizvoda');
                });
    });
});
```

Kada se obrazac za dodavanje proizvoda pokrene, JavaScript programski kod prikuplja podatke iz obrasca, stvara SOAP zahtjev i šalje ga na poslužitelj. Ako poslužitelj vrati uspješan odgovor, korisnik se obavještava o uspješnom dodavanju proizvoda i preusmjerava na početnu stranicu.

5.2. Funkcionalnosti ažuriranja proizvoda

Programski kod unutar edit_products_soap.js datoteke upravlja ažuriranjem postojećih proizvoda. Kada korisnik podnese obrazac za uređivanje, prikupljeni podaci šalju se natrag na poslužitelj putem SOAP zahtjeva za ažuriranje proizvoda.

Dio koda za dohvaćanje proizvoda je prikazan u nastavku.

```
document.addEventListener('DOMContentLoaded', function() {
// Dohvati ID proizvoda iz URL-a
const urlParams = new URLSearchParams(window.location.search);
        const productId = urlParams.get('id');
        const deleteProductButton =
document.getElementById('deleteProductButton');
        deleteProductButton.addEventListener('click', function(event)
{
        event.preventDefault();
                deleteProduct(productId);
        });
        // Dohvati podatke o proizvodu za uređivanje
        fetch('http://localhost:5500/public/soap_server.php', {
        method: 'POST',
                headers: {
                'Content-Type': 'text/xml'
```

```
                },
                body: `
            <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://localhost:5500">
                <soapenv:Header/>
                <soapenv:Body>
                    <web:getProductById>
                        <id>${productId}</id>
                    </web:getProductById>
                </soapenv:Body>
            </soapenv:Envelope>
        `
        })
        .then(response => response.text())
        .then(str => new window.DOMParser().parseFromString(str,
"text/xml"))
        .then(data => {
        // Pronađi sve elemente podataka
        const productData =
data.getElementsByTagName("return")[0].children;
                const product = {};
                // Iteriraj kroz sve elemente podataka i popuni
objekt proizvoda
                for (let i = 0; i < productData.length; i++) {
        const key =
productData[i].getElementsByTagName("key")[0].textContent;
                const value =
productData[i].getElementsByTagName("value")[0].textContent;
                product[key] = value;
        }

        // Postavi postojeće vrijednosti u polja forme
        document.getElementById('editProductName').value =
product.Naziv_proizvoda || '';

document.getElementById('editProductDescription').value =
product.Opis_proizvoda || '';
                document.getElementById('editProductPrice').value =
product.Cijena || '';
                document.getElementById('editProductQuantity').value
= product.Kolicina_na_skladistu || '';
                document.getElementById('editProductState').value =
product.Stanje || '';
        })
        .catch(error => {
        console.error('Error fetching product data:', error);

document.getElementById('editProductError').textContent = 'An error
occurred while fetching product data';
        });
        });
```

Prilikom učitavanja stranice za uređivanje proizvoda, JavaScript programski isječak dohvaća podatke o proizvodu sa poslužitelja pomoću SOAP zahtjeva i ispunjava obrazac za uređivanje.

```
// Slanje forme za ažuriranje proizvoda
document.getElementById('editProductForm').addEventListener('submit',
function(event) {
event.preventDefault();
        // Dohvati unesene vrijednosti iz forme
        const naziv_proizvoda =
document.getElementById('editProductName').value;
        const opis_proizvoda =
document.getElementById('editProductDescription').value;
        const cijena =
document.getElementById('editProductPrice').value;
        const kolicina_na_skladistu =
document.getElementById('editProductQuantity').value;
        const stanje =
document.getElementById('editProductState').value;
        // Dohvati ID proizvoda iz URL-a
        const urlParams = new
URLSearchParams(window.location.search);
        const productId = urlParams.get('id');
        // Ažuriraj proizvod putem SOAP poziva
        fetch('http://localhost:5500/public/soap_server.php', {
        method: 'POST',
                headers: {
                'Content-Type': 'text/xml'
                },
                body: `
            <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://localhost:5500">
                <soapenv:Header/>
                <soapenv:Body>
                    <web:updateProduct>
                        <id>${productId}</id>

<naziv_proizvoda>${naziv_proizvoda}</naziv_proizvoda>

<opis_proizvoda>${opis_proizvoda}</opis_proizvoda>
                        <cijena>${cijena}</cijena>

<kolicina_na_skladistu>${kolicina_na_skladistu}</kolicina_na_skladist
u>
                        <stanje>${stanje}</stanje>
                    </web:updateProduct>
                </soapenv:Body>
            </soapenv:Envelope>
        `
        })
        .then(response => response.text())
        .then(str => new window.DOMParser().parseFromString(str,
"text/xml"))
        .then(data => {
        const message =
data.getElementsByTagName("message")[0]?.textContent;
                const error =
data.getElementsByTagName("error")[0]?.textContent;
                if (error) {
        document.getElementById('editProductError').textContent =
error;
```

```
        } else {
        alert('Promjene uspješno spremljene!');
        }
```

Priloženi kod predstavlja dio koda za ažuriranje proizvoda. Kada korisnik podnese obrazac za uređivanje, podaci koju su upisani u polja se prikupljaju i šalju na poslužitelj putem SOAP zahtjeva za ažuriranje proizvoda. Ako je ažuriranje uspješno, korisnik se obavještava.

Korisnik ujedno ima i mogućnost brisanja proizvoda. Dio koda koji izvršava brisanje proizvoda je naveden u nastavku.

```
function deleteProduct(productId) {
        if (confirm("Dali ste sigurni da hoćete obrisati ovaj
proizvod?")) {
        fetch('http://localhost:5500/public/soap_server.php', {
        method: 'POST',
                headers: {
                'Content-Type': 'text/xml'
                },
                body: `
                <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://localhost:5500">
                    <soapenv:Header/>
                    <soapenv:Body>
                        <web:deleteProduct>
                            <id>${productId}</id>
                        </web:deleteProduct>
                    </soapenv:Body>
                </soapenv:Envelope>
            `
        })
                .then(response => response.text())
                .then(str => new
window.DOMParser().parseFromString(str, "text/xml"))
                .then(data => {
                const message =
data.getElementsByTagName("message")[0]?.textContent;
                        const error =
data.getElementsByTagName("error")[0]?.textContent;
                        if (error) {
                alert("Greška pri brisanju proizvoda!");
                } else {
                alert("Proizvod uspješno obrisan!");
                        window.location.href = "home.html";
                }
                })
                .catch(error => {
                console.error('Error deleting product:', error);
                        alert("An error occurred while deleting the
product!");
                });
        } }
```

Kada korisnik želi obrisati proizvod, pritiskom na tipku poslužitelju se šalje SOAP zahtjev za brisanje koji tada izvršava operaciju brisanja proizvoda od kojega je skripta prilikom učitavanja stranice dohvatila podatke.

5.3. Obrada i rukovanje zahtjeva na poslužiteljskoj strani

U *soap_server.php* se SOAP poslužitelj kreira, pokreće i dodjeljuju mu se metode za upravljanje proizvodima.

Metoda za dohvaćanje svih proizvoda koristi kod koji je naveden u nastavku.

```
    public function getProducts() {
        $sql = "SELECT * FROM proizvodi";
        $result = $this->conn->query($sql);
        $products = [];
        while ($row = $result->fetch_assoc()) {
            $products[] = $row;
        }

        return $products;
    }
```

Priloženi dio koda u nastavku predstavlja metodu koja uzima vrijednosti i dodaje proizvod sa tim vrijednostima.

```
    public function createProduct($naziv_proizvoda, $opis_proizvoda,
$cijena, $kolicina_na_skladistu, $korisnicko_ime, $stanje) {
        $query = "INSERT INTO proizvodi (Naziv_proizvoda,
Opis_proizvoda, Cijena, Kolicina_na_skladistu, Korisnicko_ime,
Stanje, Datum_objave)
                  VALUES ('$naziv_proizvoda', '$opis_proizvoda',
'$cijena', '$kolicina_na_skladistu', '$korisnicko_ime', '$stanje',
NOW())";

        if ($this->conn->query($query)) {
            return ["message" => "Product created successfully"];
        } else {
            return ["error" => "Error creating product: " . $this-
>conn->error];
        }
    }
```

Metoda koja zamjenjuje postojeće vrijednosti sa novim vrijednostima koristi kod u nastavku.

```
    public function updateProduct($id, $naziv_proizvoda,
$opis_proizvoda, $cijena, $kolicina_na_skladistu, $stanje) {
        $query = "UPDATE proizvodi
                  SET Naziv_proizvoda='$naziv_proizvoda',
Opis_proizvoda='$opis_proizvoda', Cijena='$cijena',
Kolicina_na_skladistu='$kolicina_na_skladistu', Stanje='$stanje'
                  WHERE ID_proizvoda=$id";
```

```
        if ($this->conn->query($query)) {
            return ["message" => "Product updated successfully"];
        } else {
            return ["error" => "Error updating product: " . $this-
>conn->error];
        }
    }
```

Metodu koja briše proizvod koristi priloženi kod u nastavku.

```
public function deleteProduct($id) {
        $query = "DELETE FROM proizvodi WHERE ID_proizvoda = $id";

        if ($this->conn->query($query)) {
            return ["message" => "Product deleted successfully"];
        } else {
            return ["error" => "Error deleting product: " . $this-
>conn->error];
        }
    }
```

Sve prijašne metode su dio klase pod nazivom *ProductService*. Koristeći kod u nastavku se ostvaruje kreiranje novog SOAP poslužitelja, postavljanje klase *ProductService* za rukovanje zahtjevima i na kraju pokretanje poslužitelja.

```
$server = new SoapServer(null, ['uri' => 'http://localhost:5500']);
$server->setClass('ProductService', $conn);
$server->handle();
```

Sve prijašnje navedene metode su i ujedno SOAP pristupne točke.

- getProducts - Dohvaća sve proizvode.
- createProduct - Stvara novi proizvod.
- updateProduct - Ažurira postojeći proizvod.
- deleteProduct - Briše postojeći proizvod.

6. REST i SOAP usporedba

U ovom poglavlju su prikazane glavne usporedbe REST i SOAP tehnologija koristeći implementaciju na primjeru web trgovine.

6.1. Potreban broj linija koda za implementaciju

Kodiranje SOAP-a obično zahtijeva manji broj linija koda jer koristi definicije unaprijed postavljenih poruka (npr. WSDL), što omogućuje automatski generiranje klijenta i poslužitelja.

```
$server->setClass('ProductService', $conn);
```

SOAP koristi klasu ProductService koja automatski generira pristupne točke (*endpoint*) za dohvaćanje, kreiranje, ažuriranje i brisanje proizvoda.

REST zahtijeva nešto više linija koda jer je fleksibilniji, a time je potrebno i definirati više funkcionalnosti. Svaka API pristupna točka mora biti ručno kreirana i konfigurirana.

6.2. Cijena održavanja i implementacije

Implementacija i održavanje SOAP-a obično su skuplji zbog složenosti same arhitekture. SOAP koristi XML kao standardni format za razmjenu poruka, što zahtijeva dodatnu obradu, posebno kada su podaci veliki ili kompleksni. XML je detaljan format, ali je i relativno težak za parsiranje u usporedbi s drugim formatima poput JSON-a, što znači da je potrebna veća procesorska snaga i složeniji softver za njegovo učinkovito korištenje.

REST arhitektura je dizajnirana da bude laganija i jednostavnija za implementaciju. Umjesto korištenja složenih protokola poput SOAP-a, REST se temelji na jednostavnim HTTP metodama (GET, POST, PUT, DELETE), što značajno smanjuje složenost i troškove održavanja. Podaci u REST-u mogu se prenositi u različitim formatima, ali najčešće se koristi JSON, koji je jednostavniji za čitanje i parsiranje u odnosu na XML. JSON format zauzima manje prostora i lakši je za rad s modernim aplikacijama, posebno onima koje su napisane u JavaScriptu, čime se smanjuje potreba za dodatnim procesorskim resursima i ubrzava procesiranje podataka.

6.3. Brzina izvršavanja

Koristeći metodu *performance.now()* možemo izmjeriti trajanje svakog zahtjeva. Sljedeća trajanja su izvedena tako da su zahtjevi slani na podatke iste veličine, slani su više puta i izdvojena su najkraća vremena.

Tablica 1 Duljina trajanja svakog zahtjeva

Metoda	Funkcija	SOAP vrijeme	REST vrijeme
GET	Dohvaća podatke	516ms	421ms
POST	Kreira nove podatke	43ms	29ms
PUT	Ažurira podatke	53ms	31ms
DELETE	Briše podatke	20ms	40ms

Iz tablice se može vidjeti da SOAP ima veće vrijeme izvršavanja što ga čini sporijim zbog većih paketa podataka i složenosti XML-a.

REST je brži između ostalog jer koristi lakši format poput JSON-a u komunikaciji.

6.4. Količine pogrešaka

SOAP je manje sklon pogreškama zbog svoje stroge strukture i unaprijed definiranih standarda. XML omogućava strogo provjeravanje valjanosti svake poruke na temelju unaprijed definiranih XML shema. Ova validacija osigurava da sve poruke koje se razmjenjuju između klijenta i poslužitelja prate jasno definirane standarde i strukture. Na primjer, prilikom ažuriranja proizvoda putem SOAP-a, svaki element unutar XML poruke mora biti ispravno definiran i validiran, što smanjuje rizik od pogrešaka.

REST, s druge strane, koristi fleksibilniji pristup i može biti skloniji pogreškama zbog manje stroge strukture. Iako podržava različite formate kao što su JSON, XML ili čak običan tekst, oni nemaju strogo definirane standarde za strukturu podataka kao SOAP. Ova fleksibilnost omogućuje bržu i jednostavniju integraciju, ali također može povećati rizik od grešaka. Validacija mora se ručno implementirati u svakoj aplikaciji. Na primjer, prilikom prijave ili registracije korisnika, REST API može primiti različite formate podataka, a validacija tih podataka može varirati između različitih aplikacija. Ako validacija nije pažljivo implementirana, postoji veći rizik od pogrešaka u obradi podataka, kao što su nedostajuća polja, neispravni formati podataka ili nepravilno oblikovane poruke.

6.5. Pouzdanost

SOAP je poznat po visokoj pouzdanosti zahvaljujući ugrađenim sigurnosnim i transakcijskim mehanizmima. Koristi niz protokola koji osiguravaju sigurnost, autentifikaciju i transakcijsku dosljednost, poput WS-Security. Ovi mehanizmi jamče da će poruke biti isporučene, čak i u slučaju prekida veze, te da će transakcije biti dovršene u cijelosti ili vraćene na početno stanje u slučaju pogreške.

Primjer pouzdanosti može se vidjeti prilikom pregleda i dodavanja proizvoda u sustavu. Kada se koristi SOAP, svi podaci koji se šalju ili primaju prolaze kroz stroge provjere integriteta i sigurnosti, čime se osigurava da su podaci točni, sigurno preneseni, te da se transakcije odvijaju bez gubitaka podataka.

REST se također smatra pouzdanim, ali u manje kritičnim scenarijima u usporedbi sa SOAP-om. Njegova pouzdanost dolazi iz jednostavnosti HTTP protokola, ali mu nedostaju napredni transakcijski mehanizmi poput onih u SOAP-u. Jednostavne HTTP metode koje REST koristi su dovoljno pouzdane za većinu web aplikacija, ali nemaju ugrađene transakcijske protokole za osiguravanje dosljednosti u slučaju prekida.

6.6. Performanse

SOAP ima niže performanse zbog većih količina podataka i složenije strukture. SOAP servisi koriste više resursa za dohvaćanje i obradu podataka.

REST pruža bolje performanse zbog manjih i jednostavnijih formata podataka, čime je brži u radu s korisnicima.

6.7. Neovisnost o protokolu

SOAP je neovisan o protokolu i može koristiti HTTP, TCP, SMTP, itd. SOAP bi se mogao koristiti za različite vrste komunikacije ovisno o zahtjevima za pregled ili dodavanje proizvoda.

REST je vezan isključivo uz HTTP protokol, što ga čini jednostavnijim, ali manje fleksibilnim.

6.8. Jezik kojim su podaci oblikovani

SOAP koristi XML za razmjenu podataka, što ga čini složenijim, ali sigurnijim. U primjeru s proizvodima, svi podaci su strukturirani u XML formatu.

REST koristi JSON ili XML, a često preferira JSON zbog jednostavnosti i manjeg opterećenja u komunikaciji.

6.9. Specifične karakteristike

Za SOAP se može reći da ima siguran prijenos podataka, pouzdanu razmjenu povjerljivih poruka, skalabilnost i prilagodljivost. Pogodan je za složenije sustave poput onih koji upravljaju osjetljivim podacima, npr. dodavanje i pregled proizvoda.

REST je brz, lako se implementira, ima široku upotrebu, te se često koristi za manje kritične aplikacije s brzim performansama, poput prijava i ažuriranja korisničkih podataka.

6.10. Sažeta usporedba

U tablici 2 je dana sažeta usporedba između SOAP i REST web servisa.

Tablica 2 Usporedba SOAP i REST servisa

Kategorija	**SOAP**	**REST**
Potreban broj linija koda za implementaciju	Manji broj linija koda za implementaciju	Veći broj linija koda za implementaciju
Cijena održavanja i implementacije	Veća	Manja
Brzina izvršavanja	Manja	Veća
Količine pogrešaka	Manja	Veća
Pouzdanost	Veća	Manja
Performanse	Manje	Veće
Neovisnost o protokolu	Neovisan o protokolu (HTTP, TCP, UDP, SMTP)	Ovisi o HTTP protokolu
Podaci oblikovani jezikom	XML	JSON, XML
Specifične karakteristike	-Siguran prijenos podataka -Sigurna razmjena povjerljivih poruka -Tajnost -Kredibilitet -Skalabilnost -Interoperabilnost -Prilagodljivost -Protokol	-Velike brzine obrade podataka - Laka implementacija -Lako održavanje -Korištenje URI-ja za označavanje podataka -Vrlo čitljiv -Lako ispravljanje poruka -Sve veća uporaba -Stil kojim je arhitektura opisana

7. Zaključak

U ovom radu uspoređeni su REST i SOAP web servisi kroz primjenu u web trgovini, ističući njihove ključne karakteristike, prednosti i nedostatke. REST je jednostavna, lagana i fleksibilna arhitektura koja koristi HTTP protokol i često koristi JSON za prijenos podataka. Ova jednostavnost ga čini izuzetno popularnim za web aplikacije koje zahtijevaju brze performanse i jednostavnu integraciju. S druge strane, SOAP koristi XML kao standard za razmjenu podataka te podržava više protokola, uključujući HTTP, TCP i SMTP. Njegova složenost i podrška za napredne sigurnosne i transakcijske mehanizme čine ga prikladnim za složenije aplikacije koje upravljaju osjetljivim podacima.

Struktura web trgovine prikazana u radu temelji se na SQL bazi podataka s glavnim entitetima kao što su korisnici, proizvodi i narudžbe, koji su međusobno povezani kako bi omogućili dinamičku interakciju. REST API implementiran je za osnovne funkcionalnosti poput prijave, registracije i ažuriranja korisnika, pri čemu njegova jednostavnost i široka primjena omogućuju brze performanse i lakoću održavanja. S druge strane, SOAP je korišten za složenije funkcionalnosti poput dodavanja i pregleda proizvoda, gdje je njegova pouzdanost i sigurnost ključna prednost, posebno u radu s osjetljivim poslovnim podacima.

Usporedba REST i SOAP servisa pokazuje razlike u potrebnom broju linija koda, cijeni održavanja, brzini izvršavanja i pouzdanosti. SOAP generira manje koda zahvaljujući unaprijed definiranim WSDL-om, dok REST zahtijeva ručnu konfiguraciju svake komunikacijske točke sučelja. S obzirom na cijenu, SOAP je skuplji za implementaciju i održavanje zbog složenosti XML-a i dodatnih sigurnosnih protokola, dok je REST ekonomičniji i jednostavniji za održavanje. Kada su u pitanju performanse, REST nudi brži rad zbog korištenja manjih formata podataka kao što je JSON, dok SOAP zbog veće složenosti sigurnosnih mjera ima niže performanse. Iako SOAP pruža veću pouzdanost zbog ugrađenih sigurnosnih i transakcijskih mehanizama, REST je dovoljno pouzdan za jednostavnije aplikacije s manje kritičnim podacima.

Općenito, izbor između REST-a i SOAP-a ovisi o specifičnim potrebama aplikacije. SOAP se ističe u sigurnosno zahtjevnim i transakcijski složenim sustavima, dok je REST pogodniji za brže, manje složene aplikacije. U slučaju web trgovine, SOAP se može koristiti za funkcije koje zahtijevaju visok nivo sigurnosti i pouzdanosti kao npr. obrada plaćanja, dok REST omogućava veću fleksibilnost i brzinu u operacijama poput pregleda podataka i upravljanja korisnicima. Ovisno o zahtjevima projekta, oba pristupa mogu uspješno koegzistirati u modernim aplikacijama.

8. Literatura

[1] https://www.redhat.com/en/topics/api/what-is-a-rest-api, pristupljeno 11.1.2024.

[2] https://aws.amazon.com/what-is/restful-api/ , pristupljeno 11.1.2024.

[3] https://blog.postman.com/soap-api-definition/ , pristupljeno 11.1.2024.

[4] https://blog.dreamfactory.com/when-to-use-rest-vs-soap-with-examples/,pristupljeno 13.1.2024.

[5] https://zir.nsk.hr/islandora/object/foi%3A7275/datastream/PDF/view,pristupljeno 13.1.2024.

[6] https://www.soapui.org/docs/soap-and-wsdl/authenticating-soap-requests/,pristupljeno 13.1.2024.

[7] https://www.altexsoft.com/blog/what-is-soap-formats-protocols-message-structure-and-how-soap-is-different-from-rest/, pristupljeno 19.1.2024.

[8] https://stoplight.io/api-types/soap-api#:~:text=SOAP%20is%20the%20Simple%20Object,the%20structure%20of%20its%20payloads., pristupljeno 19.1.2024.

[9] https://apitoolkit.io/blog/everything-about-soap-apis/, pristupljeno 12.1.2024.

Popis slika

Popis tablica