Automatsko kreiranje opisa web servisa

Tomašić, Matija

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:059674

Rights / Prava: Attribution 3.0 Unported/Imenovanje 3.0

Download date / Datum preuzimanja: **2024-05-18**

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matija Tomašić

AUTOMATSKO KREIRANJE OPISA WEB SERVISA

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matija Tomašić

Matični broj: 0016149486

Studij: Informacijski i poslovni sustavi

AUTOMATSKO KREIRANJE OPISA WEB SERVISA

ZAVRŠNI RAD

Mentor/Mentorica:

Izv. prof. dr. sc. Darko Andročec

Varaždin, rujan 2023.

Matija Tomašić

Izjava o izvornosti

Izjavljujem da je moj završni/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/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema se bavi problemom nedostatka strojno čitljivog opisa za većinu web servisa koji se danas koriste. Većina tih servisa su RESTfull servisi i često nemaju formalno definiranu funkcionalnost na svojoj web stranici. Cilj rada je rješavanje ovog problema kroz automatsko generiranje opisa tih servisa. Teoretski dio rada obrađuje osnove servisno-orijentirane arhitekture, web servisa, RESTfull servisa i jezika za opis funkcionalnosti web servisa. Praktični dio rada obuhvaća primjenu metoda obrade prirodnog jezika za automatsko generiranje opisa RESTfull web servisa kroz nekoliko primjera dostupnih API-ja.

Ključne riječi: web servisi, RESTful servisi, API, GPT, WSDL, OpenAPI, Swagger

Sadržaj

Sadržaj iii
1. Uvod 1
2. Servisno-orijentirana arhitektura 2
2.1. Arhitekturalni principi SOA-e 2
2.2. Karakteristike SOA 3
2.3. Dobrobiti kod korištenja SOA-e 4
2.4. Izazovi kod SOA-e 4
3. Web servisi 5
3.1. Definicija web servisa i njihova uloga u modernom razvoja aplikacija 5
3.2. Arhitektura web servisa 6
3.3. Različiti pristupi i tehnologije za implementaciju web servisa 7
3.4. RESTful servisi 8
3.4.1. Pregled principa RESTful arhitekture 8
3.4.2. Prednosti RESTful servisa 9
3.4.3. Ograničenja RESTful servisa 9
3.5. Komunikacija putem SOAP-a 10
3.5.1. Karakteristike protokola 10
3.5.2. Struktura poruke 11
3.5.3. Prednosti SOAP-a 12
3.5.4. Nedostaci SOAP-a 13
3.6. Važnost opisa web servisa 13
3.7. Pregled popularnih jezika i standarda za opis web servisa 13
3.7.1. WSDL 14
3.7.1.1. Sintaksa WSDL 15
3.7.1.2. Primjer WSDL specifikacije 15
3.7.2. UDDI 17
3.7.2.1. Vrste UDDI registara 17
3.7.2.2. Primjer UDDI specifikacije u poslovanju 17
3.7.3. WADL 19
3.7.3.1. Glavni elementi WADL-a 19
3.7.3.2. Primjer WADL specifikacije 20
3.7.4. OpenAPI specifikacija 22
3.7.4.1. Sintaksa OpenAPI 22
3.7.4.2. Primjer OpenAPI specifikacije 22
3.7.5. Standardi za semantičke servise 23
3.7.5.1. SAWSDL 24
3.7.5.2. SA-REST 24
3.7.5.3. OWL 24

3.7.5.4. RDF ... 24
4. Obrada prirodnog jezika ... 25
4.1. Važnost obrade prirodnog jezika ... 25
4.2. Tehnike obrade prirodnog jezika ... 26
4.3. Neuronske mreže ... 27
4.3.1. Tipovi neuronskih mreža ... 27
4.3.1.1. Perceptron ... 27
4.3.1.2. Više slojni perceptroni ... 28
4.3.1.3. Konvolucijske neuronske mreže ... 28
4.3.1.4. Rekurzivne neuronske mreže ... 29
4.3.1.5. Mreža dugoročne kratkoročne memorije ... 30
4.3.1.6. Generativne suparničke mreže ... 30
4.4. Duboko učenje ... 31
4.4.1. Primjena dubokog učenja ... 31
4.5. Primjena NLP u stvarnom svijetu ... 32
4.5.1. ChatGPT ... 33
4.5.1.1. Pregled svih GPT modela kroz povijest ... 34
4.5.1.2. Bing Chat ... 36
4.5.2. Bard AI ... 37
5. Implementacija aplikacije za generiranje opisa API-ja ... 38
5.1. Pregled API-ja bez strojno čitljivog opisa ... 38
5.2. Pregled API-ja opisanog OpenAPI specifikacijom ... 39
5.3. Početak implementacije aplikacije ... 41
5.3.1. Postavljanje razvojnog okruženja ... 41
5.3.1.1. Odabira alata i tehnologija za razvoj ... 42
5.3.1.2. Instalacija Node.js okruženja ... 42
5.3.1.3. Instalacija i postavljanje pripadajućih Node.js biblioteki i razvojnih okvira ... 43
5.3.2. Početna stranica ... 45
5.3.3. Dohvaćanje dokumentacije ... 45
5.3.4. Parsiranje dokumentacije ... 45
5.3.5. Implementacija GPT modela ... 47
5.3.5.1. Postavljanje autorizacijskog ključa za GPT model ... 47
5.3.6. Generiranje interaktivnog sučelja dokumentacije ... 48
5.3.6.1. Preuzimanje dokumentacije ... 49
5.4. Testiranje generiranja specifikacije nad stvarnim dokumentacijama ... 49
5.5. Testiranje ispravnosti API-ja preko SwaggerUI ... 52
5.6. Prednosti i ograničenja aplikacije ... 53
6. Zaključak ... 54
Popis literature ... 55
Popis slika ... 61
Prilozi ... 62

1. Uvod

U današnjem zahtjevnom poslovnom okruženju, teži se prema digitalizaciji i automatizaciji svega što možemo zamisliti. Grupa stručnjaka u području informatike posvećuje se pojednostavljivanju svakodnevnih poslovnih procesa kroz raznolike metode i tehnologije. Zahvaljujući njima, procesi poput plaćanja računa ili obnavljanja zaliha proizvoda, danas se mogu obaviti od kuće, na vlastitom računalu i pametnom uređaju uz pomoć nekoliko klikova. Temelj tih automatiziranih postupaka često je uspostavljen putem web servisa. Iako koncept web servisa može biti manje poznat osobama koje nisu dublje involvirane u tehničke aspekte, zapravo se radi o sredstvu komunikacije između elektroničkih uređaja putem mreže.

U ovom radu, glavni fokus nije na detaljnom razumijevanju svakodnevnih poslovnih procesa, već na naglašavanju značaja uputa, specifikacija ili opisa koji se koriste za optimalno iskorištavanje potencijala web servisa. Ove specifikacije igraju ključnu ulogu u procesu implementacije ili testiranja web aplikacija jer omogućavaju preciznu koordinaciju i učinkovitost u komunikaciji s web servisima. U teoretskom dijelu rada biti će navedena osnovna teorijska načela servisno-orijentirane arhitekture, obrade prirodnog jezika, web servisa i jezika koji služe za opisivanje funkcionalnosti web servisa, gdje će najveći fokus biti na opisivanju RESTful web servisa.

U radu će biti realizirana jednostavna aplikacija koja analizira nestandardiziranu dokumentaciju jednostavnog RESTful web servisa. Koristeći model GPT, aplikacija će generirati detaljne opise za web servis u obliku OpenAPI specifikacije. Ovaj proces ima potencijal da unaprijedi razumijevanje i korisnost web servisa te da olakša implementaciju i testiranje web aplikacija koje se oslanjaju na te servise.

2. Servisno-orijentirana arhitektura

Servisno-orijentirana arhitektura (SOA) predstavlja pristup razvoju programske infrastrukture u kojem se sam proizvod oblikuje putem sastavnih jedinica nazvanih servisi. Kroz ovakav pristup, postiže se visoka ponovna uporabljivost i međusobna povezanost. U okviru SOA-e, servisi komuniciraju putem definiranih sučelja te razmjenjuju podatke kako bi omogućili izvođenje specifičnih funkcionalnosti. Ključna prednost leži u mogućnosti višestrukog korištenja servisa, što znači da se isti servis može primijeniti unutar iste aplikacije na više mjesta ili čak podržavati različite aplikacije, bez potrebe za ponovnim pisanjem programskog koda. Fleksibilnost sustava omogućuje pozivanje, korištenje i prilagodbu servisa bez negativnog utjecaja na cjelokupnu arhitekturu [3].

2.1. Arhitekturalni principi SOA-e

U nastavku će biti navedeni ključni principi koji leže u temelju SOA-e te kako oni doprinose izgradnji sustava koji je agilan, prilagodljiv i robustan [3]:

- **Labavo povezivanje**: Servisi održavaju odnos koji smanjuje ovisnost i samo održavaju svijest jedan o drugome. Značajka koja govori da treba biti što manje ovisnosti između web usluge i klijenta koji poziva web uslugu. Dakle, ako se funkcionalnost usluge promijeni u bilo kojem trenutku, to ne bi trebalo pokvariti klijentsku aplikaciju ili zaustaviti njen rad.
- **Ugovor o usluzi**: Servisi se pridržavaju komunikacijskog sporazuma kako je zajednički definirano jednim ili više dokumenata s opisom usluge.
- **Apstrakcija usluge**: Osim onoga što je opisano u ugovoru o usluzi, servisi skrivaju logiku od vanjskog svijeta. Servisi ne bi trebali otkrivati kako izvršavaju svoju funkcionalnost. Usluga treba aplikaciji reći što radi i čemu služi, a ne kako to radi. Primjerice za *Representational state transfer model* (REST), klijent može pročitati *RESTful API Modeling Language* (RAML) ili *OpenAPI* specifikaciju usluge i može saznati što je potrebno za interakciju s uslugom.
- **Ponovna upotreba usluge**: Logika je podijeljena na servise s namjerom promicanja ponovne upotrebe. Većina razvojnih tvrtki ne vidi potrebu ponovne izgradnje programskog koda kojeg više različitih aplikacija upotrebljava, već imaju namjeru postojeći kod iskoristiti i pritom smanjiti troškove.
- **Kompozitivnost servisa**: Zbirke servisa mogu koordinirati i sastaviti u složene servise. Nije dobra praksa da se ugrađuju sve funkcije aplikacije u jedan servis, već umjesto toga rastavljaju se na module od kojih svaki ima zasebnu funkciju.

- **Autonomija usluge**: Servisi imaju kontrolu nad logikom koju enkapsuliraju. Usluga zna sve detalje o funkcionalnosti koju nudi.
- **Mogućnost otkrivanja usluge**: Usluge su osmišljene tako da budu opisane prema van kako bi se mogle pronaći i procijeniti putem dostupnih mehanizama otkrivanja.

2.2. Karakteristike SOA

Servisno orijentirana arhitektura mora imati određena svojstva koja ispunjavaju temeljne zahtjeve za obavljanje automatizacije, koja se sastoji od usluga, nad kojima su primijenjena načela dizajna usmjerenog na usluge. SOA posjeduje četiri karakteristike koje zajedno čine cjelinu koja se izdvaja od drugih arhitekturalnih modela [1, str. 61 do str. 69]:

- **Vođena poslovanjem** (eng. *business-driven*): Kada je tehnološka arhitektura usmjerena prema poslovanju, temelji se na sveobuhvatnoj poslovnoj viziji, ciljevima i zahtjevima te služi kao osnova i glavni utjecaj na oblikovanje arhitektonskog modela. To maksimizira usklađenost tehnologije i poslovanja te omogućuje tehnološkoj arhitekturi da se razvija zajedno s organizacijom u cjelini. Rezultat je neprekidno povećanje vrijednosti i trajnosti arhitekture.
- **Neutralna prema dobavljaču** (eng. *vendor-neutral*): Korištenje arhitekture usmjerene prema jednom specifičnom dobavljaču može rezultirati implementacijom koja nesvjesno nasljeđuje njegove ekskluzivne karakteristike. To može ograničiti budući razvoj arhitekture zbog nedostupnosti tehnoloških inovacija drugih dobavljača. Takva arhitektura može postati ograničenog trajanja te će je trebati zamijeniti kako bi ostala učinkovita.
- **Usmjerena na poduzeće** (eng. *enterprise-centric*): Temeljna karakteristika koja nalaže da se nad tehnološkom arhitekturom mora uspostaviti model, koji se temelji na pretpostavci da će usluge biti dijeljene s drugim dijelovima poduzeća ili će biti dio većih rješenja koja uključuju zajedničke usluge. Ovakav pristup naglašava standardizaciju dijelova arhitekture kako bi se kontinuirano poticala ponovna uporaba usluga i interoperabilnost.
- **Usmjerena na kompoziciju** (eng. *composition-centric*): Naglasak na dizajniranju softverskih programa kao ne samo resursa koji se mogu ponovno koristiti, već kao fleksibilnih resursi koji se mogu uključiti u različite strukture za razne servisno orijentirana rješenja.

2.3. Dobrobiti kod korištenja SOA-e

Primjena SOA-e donosi niz dragocjenih dobrobiti koje utječu na svaki aspekt razvoja, održavanja i prilagodbe softverskih aplikacija [35]:

- **Kreiranje skalabilnih sustava koji mogu evoluirati**: SOA omogućuje razdvajanje funkcionalnosti softvera u odvojene, autonomne usluge. To znači da se svaka usluga može razvijati, ažurirati ili zamijeniti neovisno o drugim dijelovima sustava. Ova modularnost omogućuje brži i jednostavniji razvoj, čime se olakšava skaliranje i prilagodba sustava rastućim potrebama organizacije.
- **Bolje upravljanje kompleksnim sustavima**: SOA promiče razdvajanje funkcionalnosti u specijalizirane usluge, što čini sustav manje složenim i lakšim za upravljanje. To također olakšava praćenje, dijagnostiku i otklanjanje problema u sustavu, budući da svaka usluga ima jasno definiranu funkcionalnost i odgovornost.
- **Neovisnost o pojedinoj platformi**: SOA omogućuje razvijanje usluga neovisnih o specifičnoj tehnološkoj platformi. To omogućuje organizacijama da iskoriste različite tehnologije i dobavljače za različite dijelove sustava, čime se smanjuje ovisnost o jednom dobavljaču ili tehnologiji.

2.4. Izazovi kod SOA-e

Usprkos mnogim prednostima koje donosi SOA, ona istovremeno stavlja pred organizacije niz izazova koji zahtijevaju temeljito planiranje i pažljivo vođenje [36]:

- **Upravljanje servisima**: Nedovoljna pažnja posvećena upravljanju servisima može dovesti do problema s performansama, skalabilnosti, pouzdanošću i održavanjem. Potrebno je pažljivo nadzirati i upravljati uslugama kako bi se osigurala njihova optimalna izvedba i kontinuirana dostupnost.
- **Pružanje odgovarajuće sigurnosti za pojedine usluge**: Svaka usluga u SOA arhitekturi treba biti adekvatno zaštićena zato što može imati različite sigurnosne zahtjeve, ovisno o prirodi i osjetljivosti podataka koje obrađuje. Implementacija odgovarajućih sigurnosnih mehanizama za svaku uslugu spriječila bi sigurnosne propuste i neovlašteni pristup, te time bi se osigurala cjelovitost i povjerljivost podataka.

3. Web servisi

U današnjem digitalnom svijetu, web servisi su postali nezamjenjiv temelj moderne tehnologije. Oni predstavljaju ključni mehanizam koji omogućuje aplikacijama da se povezuju, surađuju i razmjenjuju informacije preko interneta. Bez web servisa, mnoge od naših omiljenih online usluga i funkcionalnosti ne bi bilo moguće ostvariti. Ovi moćni alati omogućavaju nam da integriramo različite tehnološke platforme, bez obzira na to jesu li razvijene u različitim programskim jezicima, koriste različite baze podataka ili su smještene na različitim geografskim lokacijama. Zahvaljujući web servisima, aplikacije mogu komunicirati međusobno, razmjenjivati podatke, pristupati funkcionalnostima drugih aplikacija i pružati korisnicima bogato i povezano iskustvo.

3.1. Definicija web servisa i njihova uloga u modernom razvoja aplikacija

Web servis je softverski modul dostupan putem mreže, poput interneta, koji obavlja zadatke, rješava probleme ili provodi transakcije u ime korisnika ili aplikacije. Web servisi čine distribuiranu računalnu infrastrukturu koju čine mnogi različiti međusobno povezani moduli aplikacija koji pokušavaju komunicirati putem privatnih ili javnih mreža (uključujući internet i web) kako bi virtualno formirali jedan logički sustav. Web servis može imati biti:

- **samodostatan poslovni zadatak**, poput usluge povlačenja ili uplate sredstava
- **cjelovit poslovni proces**, kao što je automatizirana kupnja uredskog materijala
- **aplikacija**, poput aplikacije za upravljanje i obrađivanje polica osiguranja klijenata ili aplikaciju koju tvrtka koristi za prognozu potražnje njihovih proizvoda ili usluga, te obnavljanje zaliha
- **resurs osposobljen za pružanje usluge** poput pristupa bazi podataka s medicinskim zapisima pacijenata.

Web servisi mogu varirati u funkciji od jednostavnih zahtjeva (npr. provjera kreditne sposobnosti i ovlaštenja, upit cijena, provjera statusa inventara ili vremenska prognoza) do potpunih poslovnih aplikacija koje pristupaju i kombiniraju informacije iz više izvora, poput sustava za posredovanje osiguranja, izračuna odgovornosti osiguranja, automatiziranog planiranja putovanja ili sustava praćenja pošiljki [2, str. 5].

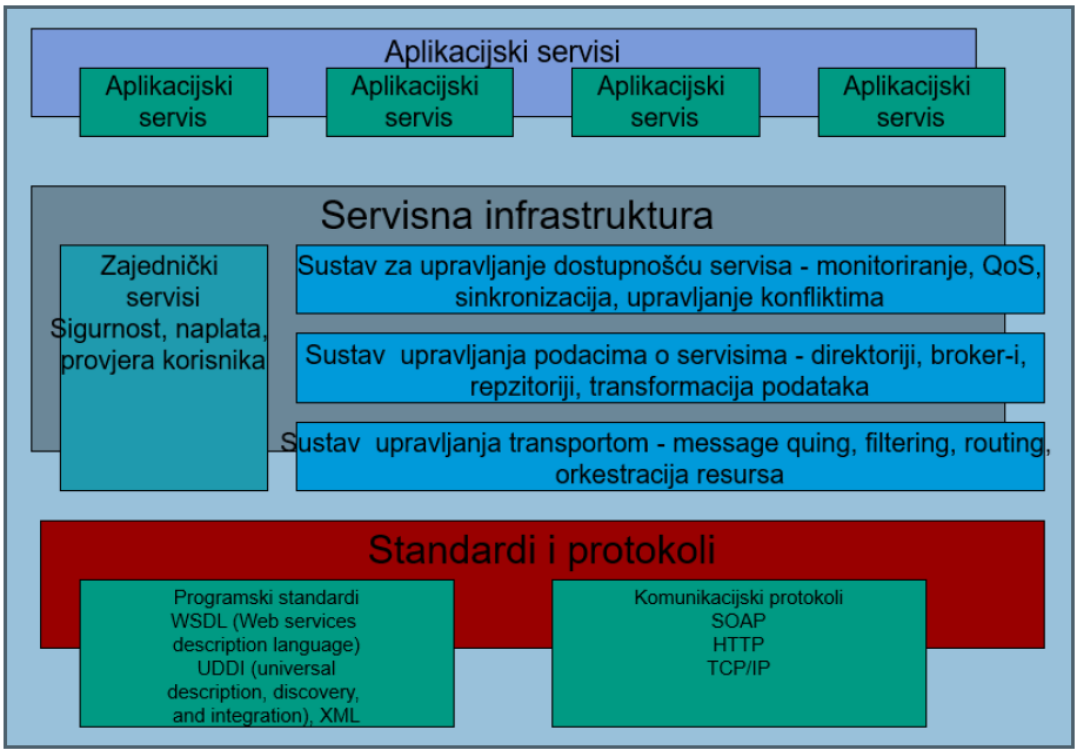
3.2. Arhitektura web servisa

S ciljem osiguranja potrebne funkcionalnosti, zaštite i visoke kvalitete, arhitektura web servisa se temelji na sljedećim slojevima i komponentama [32], [37], [38]:

1. **Aplikacijski sloj**: Ovaj sloj sastoji se od nekoliko aplikacijskih servisa koji pružaju različite funkcionalnosti i usluge korisnicima. Svaki servis može obavljati određenu poslovnu logiku, a moguće je imati servise kao što su upravljanje korisnicima, obrada plaćanja, provjera korisnika, itd. Ovi servisi su međusobno neovisni i komuniciraju putem definiranih sučelja.
2. **Servisna arhitektura**: Ovaj sloj se sastoji od nekoliko komponenata koje omogućuju upravljanje servisima, sigurnošću, dostupnošću i manipulacijom podataka.
    a) Zajednički servisi, sigurnost, naplata, provjera korisnika: Ove komponente pružaju zajedničke usluge koje se mogu dijeliti između različitih aplikacijskih servisa. Na primjer, sigurnosni servis osigurava autentikaciju i autorizaciju korisnika, naplatni servis omogućuje naplatu usluga, dok provjera korisnika provodi validaciju i autentikaciju korisnika.
    b) Sustav za upravljanje dostupnošću servisa: Ova komponenta odnosi se na monitoring i upravljanje kvalitetom usluge (eng. *Quality of Service,* kraće QoS) servisa. To uključuje nadzor nad dostupnošću servisa, praćenje performansi i reagiranje na prekide ili degradaciju usluge. Sinkronizacija se odnosi na održavanje konzistentnosti između više instanci istog servisa.
    c) Sustav upravljanja podacima o servisima: Ova komponenta bavi se upravljanjem informacija o dostupnim servisima. To može uključivati direktorije, repozitorije i brokerske komponente koje olakšavaju pronalaženje i povezivanje sa servisima. Također, transformacija podataka omogućuje pretvaranje formata podataka između različitih servisa.
    d) Sustav upravljanja transportom: Ova komponenta brine se o komunikaciji između servisa. To uključuje aspekte kao što su message queuing (slanje i primanje poruka), filtriranje i usmjeravanje poruka te orkestracija resursa kako bi se osiguralo da poruke stignu do pravih mjesta.
3. **Standardi i protokoli**:
    a) Programski standardi (*WSDL*, *UDDI*, *XML*): *Web Services Description Language* (kraće *WSDL*) koristi se za opisivanje funkcionalnosti servisa putem sučelja i metoda koje pruža. *Universal Description, Discovery, and Integration* (kraće *UDDI*) omogućuje registraciju i pretraživanje web servisa. *eXtensible*

Markup Language (kraće *XML*) koristi se za strukturiranje i razmjenu podataka između različitih servisa.

b) Komunikacijski protokoli (*SOAP*, *HTTP*, *TCP/IP*): *Simple Object Access Protocol* (kraće *SOAP*) je protokol koji omogućuje komunikaciju između različitih servisa putem *XML* poruka. *Hypertext Transfer Protocol* (kraće *HTTP*) koristi se za prijenos podataka putem weba. *Transmission Control Protocol/Internet Protocol* (kraće *TCP/IP*) je osnovni komunikacijski protokol koji omogućuje povezivanje na internet i razmjenu podataka.



Slika 1: Arhitektura web servisa [32]

3.3. Različiti pristupi i tehnologije za implementaciju web servisa

Ovisno o specifičnim zahtjevima i potrebama projekta, razvojni timovi biraju pristup i tehnologiju za implementaciju web servisa. Izbor pravog pristupa treba zadovoljavati sljedeće karakteristike:

- **Skalabilnost**: Od web servisa se očekuje da se nosi sa velikim brojem zahtjeva.

- **Pouzdanost**: U svakom trenutku, web servisi moraju biti dostupni, mora biti implementiran mehanizam upravljanja greški.
- **Visoka performansa**: Brzina i dobar odziv osiguravaju dobro korisničko iskustvo, optimiziranje performansi i baze podataka pomažu u takvom pothvatu.
- **Sigurnost**: Web servisi trebaju biti zaštićeni od sigurnosnih prijetnji, kao što su hakiranje i zlonamjerni napadi. Kako bi se to spriječilo, potrebno je implementirati autentifikaciju, autorizaciju te enkripciju podataka.

U današnjem digitalnom svijetu najzastupljeniji pristupi razvoju web servisa su *RESTful* i *SOAP* web servisi. Iako oba pristupa imaju za cilj omogućiti komunikaciju između klijenata i servera putem interneta, postoje značajne razlike u njivom pristupu i implementaciji koje će kasnije biti objašnjene.

3.4. RESTful servisi

RESTful servisi su web servisi koji se temelje na arhitekturi *REST* (eng. *Representational State Transfer*). *RESTful* web servisi koriste *HTTP* protokol u klijent-server komunikaciji i standardne *HTTP* metode (*GET, POST, PUT, DELETE*) kako bi omogućili komunikaciju. Komunikacijom se ostvaruje manipulacija resursima na serveru koji se najčešće vraćaju u *XML* i *JavaScript Object Notation* (kraće *JSON*) obliku.

RESTful web servisi su popularni zbog svoje skalabilnosti, jednostavnosti i fleksibilnosti. Omogućavaju web aplikacijama izgrađenim pomoću različitih programskih jezika da komuniciraju međusobno bez obzira na okruženje na kojem su smješteni, bilo to *Windows* ili *Linux*. Također, s porastom popularnosti mobilnih uređaja u odnosu na stolna računala, korištenje *REST-a* olakšava komunikaciju između mobilnih aplikacija i standardnih web aplikacija, bez potrebe za dodatnim naporom za prilagodbu uređajima. Budući da moderne aplikacije često moraju biti kompatibilne s oblakom, *RESTful* pristup je logičan izbor za programiranje web servisa, jer oblak temelji svoje arhitekture na principima REST-a [4].

3.4.1. Pregled principa RESTful arhitekture

RESTful arhitektura za izgradnju web servisa se temelji na nekoliko ključnih principa. U nastavku je pregled tih principa [6], [39]:

- **Resursno usmjeren pristup**: Resursi su ključni elementi *RESTful* arhitekture. Takav pristup naglašava da sve komponente sustava predstavljaju resurse s jedinstvenim identifikatorom koji se zove *Uniform Resource Identifier* (kraće *URI*). Resursi u praksi

mogu predstavljati bilo koju vrstu informacije i funkcionalnost, kao što su prikaz i upravljanje korisničkim računima, proizvodima, narudžbama i slično.

- ***HTTP* metode**: Jedan od najvažnijih elemenata komunikacije kod *RESTful* web servisa. *HTTP* metode (*GET, POST, PUT, DELETE*) služe za manipulaciju sa resursima na serveru. U praksi, *GET* metoda se koristi za dohvaćanje resursa, *POST* metoda se koristi za stvaranje novog resursa, *PUT* metoda se koristi za ažuriranje postojećeg resursa, a *DELETE* metoda se koristi za brisanje resursa [5].
- **Rezultat** (eng. *Representation*): Kod rezultata ili odgovora u komunikaciji između klijenta i servera, *RESTful* servisi koriste jednostavne formate u obradi informacija, poput *XML* i *JSON* formata. U praksi se najčešće preferira *JSON* zbog svoje jednostavnosti i lakše čitljivosti.
- **Bez stanja** (eng. *Stateless*): RESTful web servisi su bez stanja, što znači da se ne pohranjuju informacije između dvije uzastopne komunikacije. Svaki zahtjev prema serveru ne ovisi o prethodnim zahtjevima, već samo o trenutnom. Primjerice kod svakog zahtjeva autentifikacije ili autorizacije potrebno je priložiti povjerljive podatke.

3.4.2. Prednosti RESTful servisa

U svijetu razvoja web aplikacija i sustava, mnogi timovi se oslanjaju na RESTful servise zbog njihovih brojnih prednosti, uključujući:

- **Jednostavnost**: Razumijevanje na koji način pristupiti komunikaciji između servera i klijenta, poput dohvaćanja podataka putem URI i metode je jednostavno.
- **Neovisnost o jeziku i platformi razvoja**: Mogu se napisati u bilo kojem programskom jeziku i mogu se održavati na bilo kojoj platformi.
- **Fleksibilnost formata vraćenih podataka**: Nije potrebno dodatno parsirati vraćeni odgovor za razliku kod drugih servisa, već je dovoljno da se odgovor vrati u formatu u kojem se želi i u kojem je to moguće, najčešće su to XML i JSON formati

3.4.3. Ograničenja RESTful servisa

Iako je RESTful arhitektura popularna zbog svojih brojnih prednosti, postoje i neka ograničenja koja mogu utjecati na njenu učinkovitost. U nastavku su navedena najpoznatija ograničenja [43]:

- **Nedostatak sigurnosti**: *RESTful* servisi su osjetljivi na sigurnosne prijetnje poput *Distributed Denial-of-Service* (kraće *DDoS*) napada, *Structured query language* (kraće SQL) injekcija i *Cross-site scripting* (kraće XSS) napada. Stoga se moraju primjenjivati odgovarajuće sigurnosne mjere, poput autentifikacije, autorizacije, enkripcije i drugih.

- **Nedostatak standardizacije**: *RESTful* servisi nemaju strogo definirane standarde poput *SOAP-a*, što može dovesti do različitih interpretacija i različitih implementacija *RESTful* servisa od strane različitih programera ili razvojnih timova. Bez formalnih standarda, postoji veća fleksibilnost i sloboda u izgradnji *RESTful* servisa, ali istovremeno može dovesti do problema s kompatibilnošću i interoperabilnošću ako se ne prate zajednički dogovoreni obrasci i prakse.
- **Nedostatak formalnog opisa**: *RESTful* servisi nemaju strogu specifikaciju koja bi definirala najbolju praksu i postupak za implementaciju *RESTful* servisa, ali se može koristiti *OpenAPI* specifikacija za dokumentiranje.

3.5. Komunikacija putem SOAP-a

SOAP je *XML*-bazirani protokol za pristup web servisima preko *HTTP-a*. *SOAP* je protokol ili drugim riječima, definicija kako web servisi međusobno komuniciraju s klijentskim aplikacijama koje ih pozivaju. Protokol je razvijen kao posrednički jezik kako bi aplikacije izgrađene na različitim programskim jezicima mogle lako međusobno komunicirati i izbjeći ekstremne napore u razvoju.

U današnjem svijetu postoji veliki broj aplikacija izgrađenih na različitim programskim jezicima, što čini razmjenu podataka između njih složenom. Za rješavanje ove kompleksnosti koristi se *XML* kao posrednički jezik za razmjenu podataka. *XML* je razumljiv svakom programskom jeziku i koristi se kao temeljni medij za razmjenu podataka. No, nedostaju standardne specifikacije za upotrebu *XML-a* u svim programskim jezicima. Tu ulazi *SOAP* softver, dizajniran za rad s *XML-om* preko *HTTP-a* i s nekom vrstom univerzalne specifikacije za razmjenu podataka između aplikacija [7].

3.5.1. Karakteristike protokola

SOAP je mrežni aplikacijski protokol koji se koristi za prijenos poruka između servisnih instanci opisanih sučeljima *WSDL-a*. *SOAP* poruke koriste različite protokole poput *HTTP-a* za prijenos poruka i pronalaženje udaljenih sustava povezanih s interaktivnim web uslugama. *SOAP* opisuje kako se poruka formatira, ali ne specificira kako se dostavlja. Da bi se postigao taj cilj, poruka se mora umetnuti u protokol na razini prijenosa. *HTTP* je najčešće korišteni transportni protokol, ali mogu se koristiti i drugi protokoli, poput *Simple Mail Transfer Protocol* (kraće *SMTP*), *File Transfer Protocol* (kraće *FTP*) ili *Remote Method Invocation* (kraće *RMI*).

SOAP igra ulogu mehanizma povezivanja između dvije komunicirajuće točke. *SOAP* točka završetka je jednostavno *URL* zasnovan na *HTTP-u* koji identificira cilj za poziv metode. Cilj *SOAP-a* je omogućiti fleksibilno povezivanje. Na primjer, određena web usluga može

pružiti dva povezivanja. Klijent može poslati *SOAP* zahtjev koristeći ili *HTTP* ili e-mail putem *SMTP-a*. Važno je shvatiti da vrsta korištenog povezivanja ne utječe na oblikovanje *SOAP* poruke.

U najnovijoj verziji (*SOAP* verzija 1.2), *SOAP* dodaje jasna pravila za podršku novim transportnim protokolima poput *SMTP-a* ili F*TP-a*, čineći ga globalnim protokolom razmjene. Također, *SOAP* uvodi koncept posrednika (staje između *SOAP* poziva i krajnje točke koja ne utječe na sadržaj poruke, korisno za usmjeravanje informacija). Ova ideja o krajnjoj točki znači da *SOAP* poruka može poslati informaciju cijelim nizom posrednika do konačnog primatelja. Informacije o raznim posrednicima i podaci posvećeni njima nalaze se u zaglavlju *SOAP* poruke. Informacije u tijelu *SOAP* poruke namijenjene su krajnjem korisniku poruke [2, od str. 121 do str. 149].

3.5.2. Struktura poruke

SOAP poruka se sastoji od 4 dijela: omotnice, zaglavlja, tijela poruke i dijela nazvanog greška [32], [42]:

- **Omotnica** (eng. *envelope*): Vanjski omotač koji okružuje sve dijelove *SOAP* poruke, uključujući zaglavlje, tijelo i dio rezerviran za grešku. Envelope definira početak i kraj *SOAP* poruke, te sadrži informacije o verziji *SOAP* protokola koji se koristi.
- **Zaglavlje** (eng. *header*): Sadrži dodatne informacije o poruci, poput informacija o sigurnosti, identitetu i adresi primatelja. Zaglavlje je opcionalno i može biti prazno.
- **Tijelo** (eng. *body*): Sadrži stvarni sadržaj poruke u XML formatu. Tijelo je obvezni dio poruke i sadrži poziv metode (eng. *Remote procedure call - RPC*) ili informacije potrebne za obradu zahtjeva i odgovora.
- **Greška** (eng. *fault*): Ovaj dio je opcionalan i sadrži informacije o greškama koje se dogode tijekom obrade poruke. Ako se dogodi greška, *SOAP* poslužitelj šalje poruku s greškom u tijelu poruke, a klijent može obraditi grešku i poslati novu poruku s ispravkom.

Slika 2: Struktura SOAP poruke [32]

3.5.3. Prednosti SOAP-a

Kao i sa svakim drugim protokolom, nekoliko aspekata korištenja *SOAP-a* može se smatrati prednostima, dok se drugi aspekti mogu smatrati ograničenjima. Glavne prednosti *SOAP-a* su [2, str. 143]:

- **Jednostavnost**: *SOAP* je jednostavan jer se temelji na *XML-u*, koji je visoko strukturiran i lako se parsira.
- **Prenosivost**: *SOAP* je prenosiv bez ikakvih ovisnosti o temeljnoj platformi kao što su problemi s poretkom bajtova ili širinom strojne riječi. Danas postoje *XML* parseri za praktički svaku platformu, od velikih računalnih sustava do uređaja s malim resursima.
- **Prijateljstvo prema zaštitnim zidovima** (eng. *Firewall*): Slanje podataka putem *HTTP-a* znači da ne samo da je mehanizam dostave široko dostupan, već i da *SOAP* može proći kroz zaštitne zidove koji predstavljaju probleme drugim metodama.
- **Korištenje otvorenih standarda**: *SOAP* koristi otvoreni standard *XML-a* za oblikovanje podataka, što ga čini lako proširivim i dobro podržanim.
- **Interoperabilnost**: *SOAP* se temelji na otvorenim, umjesto na *vendor-specifičnim* tehnologijama i olakšava pravu distribuiranu interoperabilnost i slabo povezane aplikacije.
- **Univerzalna prihvaćenost**: *SOAP* je najšire prihvaćeni standard u domeni komunikacije poruka.

- **Otpornost na promjene**: Promjene u *SOAP* infrastrukturi vjerojatno neće utjecati na aplikacije koje koriste protokol, osim ako se značajno ne promijene specifikacije serijalizacije u *SOAP* specifikaciji.

3.5.4. Nedostaci SOAP-a

Uz svoje prednosti, postoji nekoliko aspekata *SOAP*-a koji se mogu smatrati nepovoljnim. To uključuje sljedeće [2, str. 144]:

- ***SOAP* je prvotno bio povezan s *HTTP-om*** i to je nametnulo arhitekturu zahtjev/odgovor koja nije bila primjerena za sve situacije. *HTTP* je relativno spor protokol i naravno, performanse *SOAP-a* su time bile pogođene. Najnovija verzija *SOAP* specifikacije popušta ovo ograničenje.
- ***SOAP* je bez stanja.** Beznačajna priroda *SOAP-a* zahtijeva da zahtijevajući aplikacija mora ponovno uspostaviti svoj identitet drugim aplikacijama kada je potrebno više veza, kao da nikada prije nije bila povezana. Održavanje stanja veze je poželjno kad više web usluga djeluje u kontekstu poslovnih procesa i transakcija.
- ***SOAP* serializira po vrijednosti i ne podržava serializaciju putem reference**. Serializacija po vrijednosti zahtijeva da će tijekom vremena više kopija objekta sadržavati informacije o stanju koji nisu sinkronizirani s drugim odvojenim kopijama istog objekta. To znači da trenutno nije moguće da *SOAP* upućuje ili pokazuje na neki vanjski izvor podataka (u obliku objektne reference).

3.6. Važnost opisa web servisa

Opis web servisa je strojno razumljiva specifikacija koja opisuje strukturu, operativne karakteristike i nefunkcionalna svojstva web usluge. Kao što je spomenuto ranije, opis dovoljno dobro izolira sve tehničke detalje, npr. elemente specifične za stroj i implementacijski jezik, od aplikacije ili klijenta koja šalje zahtjev pružatelju usluge [2, str. 148]. Opisi web servisa igraju ključnu ulogu u olakšavanju integracije, razumijevanja i upotrebe web servisa, pružajući standardizirane informacije koje su bitne za programere i druge korisnike servisa.

3.7. Pregled popularnih jezika i standarda za opis web servisa

Kao što je spomenuto, opis web servisa ima zadatak na najbolji način opisati operativne karakteristike i nefunkcionalne dijelove usluge gdje se u pravilu svaka usluga na neki način razlikuje pa posljedično odabir jezika za opisivanje web servisa ovisi o nekoliko faktora:

- **Potrebe projekta**: Važno je uzeti u obzir potrebe projekta i specifičnosti web servisa koje treba opisati. Na primjer, ako je potrebno opisati *RESTful* web servise, jezik poput *OpenAPI* ili *RAML-a* može biti prikladan. Ako je potrebno opisati *SOAP* web servise, *WSDL* može biti bolji izbor.
- **Kompatibilnost s postojećom infrastrukturom**: Ako postojeća infrastruktura ili alati u projektu podržavaju određeni jezik za opis web servisa, može biti praktično koristiti taj jezik kako bi se osigurala kompatibilnost i lakša integracija.
- **Aktivna podrška i zajednica**: Popularniji jezici imaju veću zajednicu korisnika pa tako mogu pružiti veći broj rješenja za probleme koji se mogu pojaviti tijekom faze opisivanja web servisa.
- **Fleksibilnost i buduće potrebe**: Važno je razmotriti fleksibilnost jezika za opis web servisa i njegovu sposobnost da podržava buduće potrebe projekta. Jezik koji omogućava lako proširivanje i prilagodbu može biti koristan ako se očekuje da će se zahtjevi i funkcionalnosti web servisa mijenjati tijekom vremena.

3.7.1. WSDL

Kako biste implementirali *SOAP* protokol s određenom web uslugom, potrebna je dokumentacija koja objašnjava strukturu *SOAP* poruka, koji će se protokol koristiti prilikom transporta podataka (npr. *HTTP* ili *SMTP*), izložene operacije zajedno s njihovim parametrima u strojno razumljivom standardnom formatu te internetsku adresu web usluge u pitanju. *WSDL* olakšava iskorištavanje prednosti *SOAP-a* pružajući način za suradnju pružatelja i korisnika takvih usluga.

WSDL je jezik za prikaz usluge koji se koristi za opisivanje pojedinosti potpuno izloženih sučelja web usluga i time omogućuje pristupanje web usluzi. Kroz opis usluge pružatelj usluge može izložiti sve specifikacije za pozivanje određene web usluge tražitelju usluge. Na primjer, niti korisnik usluge niti pružatelj usluge ne moraju biti svjesni tehničke infrastrukture, programskog jezika ili okvira za distribuirane objekte trećih strana. Iako je *WSDL* dizajniran tako da može izražavati vezivanja za protokole osim *SOAP-a*, naša je glavna briga u ovom poglavlju *WSDL* u vezi s *SOAP-om* preko *HTTP-a*. *WSDL* je specifikacija sheme temeljena na *XML-u* koja opisuje javno sučelje web usluge. To javno sučelje može uključivati operativne informacije vezane uz web uslugu kao što su sve javno dostupne operacije, informacije o tipu podataka poruka, informacije o vezivanju za određeni transportni protokol koji će se koristiti i informacije o adresi za lociranje web usluge. *WSDL* omogućuje specifikaciju usluga putem *XML* dokumenata za prijenos putem *SOAP-a*.

Iako je opis usluge web usluge u *WSDL-u* napisan isključivo s gledišta web usluge (ili pružatelja usluge koji objavljuje tu uslugu), inherentno je namijenjen i ograničava i pružatelja

usluge i tražitelja usluge koji koristi tu uslugu. To znači da *WSDL* predstavlja "ugovor" između tražitelja usluge i pružatelja usluge, na isti način kao što sučelje u objektno orijentiranom programskom jeziku, primjerice *Java*, predstavlja ugovor između klijentskog koda i stvarnog objekta. Osnovna razlika je što je *WSDL* neovisna o platformi i jeziku te se koristi uglavnom (ali ne isključivo) za opisivanje *SOAP* omogućenih usluga. U osnovi, *WSDL* se koristi za precizno opisivanje što usluga radi, tj. operacije koje usluga pruža, gdje se nalazi, tj. pojedinosti protokola specifične za adresu, poput *URL-a*, i kako je pozvati, tj. pojedinosti o formatima podataka i protokolima potrebnim za pristup operacijama usluge [2, str. 148 i str. 149].

3.7.1.1. Sintaksa WSDL

Iako WSDL dokument temeljno obuhvaća types, operations i binding elemente, važno je istaknuti da sadrži i druge važne komponente [44]:

- ***definitions***: Korijenski element koji sadrži sve druge dijelove WSDL dokumenta.
- ***types***: Definira tipove podataka korištenih u porukama.
- ***message***: Definira strukturu poruka koje se razmjenjuju između servisa i klijenta.
- ***portType***: Definira operacije koje servis pruža, zajedno s njihovim ulaznim i izlaznim porukama.
- ***operation***: Element koji povezuje poslovnu logiku servisa s definicijom sučelja koje klijent koristi za komunikaciju s tim servisom.
- ***binding***: Definira vezanje za određeni komunikacijski protokol (npr. *SOAP*) i stil komunikacije (npr. *RPC*).
- ***service***: Definira servis i pridružuje mu određeni *port* i *binding*.

3.7.1.2. Primjer WSDL specifikacije

U nastavku se nalazi primjer *HelloService.wsdl* datoteke sa prethodno spomenutim komponentama [14]:

```
<definitions name = "HelloService"
   targetNamespace = "http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns = "http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns = "http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns:xsd = "http://www.w3.org/2001/XMLSchema">

   <message name = "SayHelloRequest">
      <part name = "firstName" type = "xsd:string"/>
   </message>

   <message name = "SayHelloResponse">
```

```
            <part name = "greeting" type = "xsd:string"/>
         </message>

         <portType name = "Hello_PortType">
            <operation name = "sayHello">
               <input message = "tns:SayHelloRequest"/>
               <output message = "tns:SayHelloResponse"/>
            </operation>
         </portType>

         <binding name = "Hello_Binding" type = "tns:Hello_PortType">
            <soap:binding style = "rpc"
               transport = "http://schemas.xmlsoap.org/soap/http"/>
            <operation name = "sayHello">
               <soap:operation soapAction = "sayHello"/>
               <input>
                  <soap:body
                     encodingStyle                                          =
"http://schemas.xmlsoap.org/soap/encoding/"
                     namespace = "urn:examples:helloservice"
                     use = "encoded"/>
               </input>

               <output>
                  <soap:body
                     encodingStyle                                          =
"http://schemas.xmlsoap.org/soap/encoding/"
                     namespace = "urn:examples:helloservice"
                     use = "encoded"/>
               </output>
            </operation>
         </binding>

         <service name = "Hello_Service">
            <documentation>WSDL File for HelloService</documentation>
            <port binding = "tns:Hello_Binding" name = "Hello_Port">
               <soap:address
                  location = "http://www.examples.com/SayHello/" />
            </port>
         </service>
      </definitions>
```

3.7.2.UDDI

Universal Description, Discovery, and Integration (kraće UDDI) je standard za opisivanje, objavljivanje i pronalaženje informacija o web servisima. *UDDI* predstavlja registar za tvrtke koje pružaju web servise kako bi se predstavile i pronašle partnere na internetu. *UDDI* standard uključuje skup specifikacija, koje se zajedno objedinjuju u distribuiranu registarsku uslugu za web servise. Temelji se na specifikaciji u obliku *XML*-a i platformski je neovisan okvir koji koristi *WSDL* za opis sučelja web usluga. Predstavlja otvoren okvir širokog industrijskog zahvata jer omogućava svim vrstama tvrtki da se prikažu na internetu, otkriju i komuniciraju jedni s drugima [45].

UDDI se često uspoređuje s tradicionalnim telefonskim imenikom koji ima bijele, žute i zelene stranice [46]:

- **Bijele stranice** sadrže osnovne informacije o tvrtki i njenom poslovanju poput naziva tvrtke, adrese, kontakt broja i identifikatora poslovanja koji služi kako bi drugi otkrili našu uslugu.
- **Žute stranice** sadrže više detalja o tvrtki. To podrazumijeva opise o vrstama elektroničkih mogućnosti koje tvrtka može ponuditi kod mogućeg poslovanja. Žute stranice koriste opće prihvaćene industrijske kategorizacijske sheme, industrijske kodove, proizvodne kodove, poslovne identifikacijske kodove i slično kako bi tvrtkama olakšale pretraživanje oglasa i pronalaženje onoga što točno žele.
- **Zelene stranice** sadrže tehničke informacije o web usluzi koje omogućavaju nekome da se poveže s web uslugom nakon što je pronađena. To uključuje različita sučelja, *URL* lokacije, informacije vezane uz izvođenje web usluge.

3.7.2.1. Vrste UDDI registara

Dostupna su dva tipa registara: privatni i javni. Privatni registri namijenjeni su unutarnjoj uporabi tvrtke. Tvrtke ih koriste kako bi unutar svoje organizacije organizirale i katalogizirale svoje interne web usluge, testirale ih i omogućile razmjenu informacija između različitih internih timova ili odjela. Koriste se za unutarnje potrebe i ne omogućuju pristup vanjskim korisnicima. Podaci o internim uslugama su zaštićeni od neovlaštenog pristupa. Tvrtka ima potpunu kontrolu nad svojim internim registrom, što omogućuje prilagodbu i upravljanje prema svojim potrebama [45].

3.7.2.2. Primjer UDDI specifikacije u poslovanju

Kod ispod predstavlja informacije o poslovnom subjektu u *UDDI* registru. Svaki poslovni subjekt ima *businessKey* (jedinstveni ključ), definiranog operatora te autorizirano ime. Pod *discoveryURLs* se nalazi *URL* za otkrivanje ovog subjekta, a pod *name* je naziv subjekta,

u ovom slučaju *XMethods*. *Description* opisuje subjekt kao resursno mjesto za web usluge na engleskom jeziku. U odjelu *contacts* navedeni su kontakti za ovaj subjekt, uključujući ime osobe (*personName*), telefon i e-mail adresu. *businessServices* dio sadrži poslovne usluge koje nudi ovaj subjekt. Svaka usluga ima *serviceKey* (ključ) i ime, kao i opis na engleskom jeziku. U ovom primjeru, usluga se naziva *XMethods Delayed Stock Quotes* i opisuje se kao usluga za dohvaćanje cijena dionica s kašnjenjem od 20 minuta. Svaka usluga može imati više *bindingTemplates* koje definiraju način povezivanja s uslugom. Svaki *bindingTemplate* ima svoj ključ (*bindingKey*), opis na engleskom jeziku te *accessPoint* koji specificira *URL* na kojem je usluga dostupna [47].

```
      <businessEntity businessKey="ba744ed0-3aaf-11d5-80dc-002035229c64"
                      operator="www.ibm.com/services/uddi"
                      authorizedName="0100001QS1">
        <discoveryURLs>
          <discoveryURL
useType="businessEntity">http://www.ibm.com/services/uddi/uddiget?businessK
ey=BA744ED0-3AAF-11D5-80DC-002035229C64</discoveryURL>
        </discoveryURLs>
        <name>XMethods</name>
        <description xml:lang="en">Web services resource site</description>
        <contacts>
          <contact useType="Founder">
            <personName>Tony Hong</personName>
            <phone useType="Founder" />
            <email useType="Founder">thong@xmethods.net</email>
          </contact>
        </contacts>
        <businessServices>
          <businessService              serviceKey="d5921160-3e16-11d5-98bf-
002035229c64"
                           businessKey="ba744ed0-3aaf-11d5-80dc-
002035229c64">
            <name>XMethods Delayed Stock Quotes</name>
            <description     xml:lang="en">20-minute     delayed     stock
quotes</description>
            <bindingTemplates>
              <bindingTemplate          bindingKey="d594a970-3e16-11d5-98bf-
002035229c64"
                               serviceKey="d5921160-3e16-11d5-98bf-
002035229c64">
                <description xml:lang="en">SOAP binding for delayed stock
quotes service</description>
```

```
                <accessPoint
URLType="http">http://services.xmethods.net:80/soap</accessPoint>
                <tModelInstanceDetails>
                  <tModelInstanceInfo    tModelKey="uuid:0e727db0-3e14-11d5-
98bf-002035229c64" />
                </tModelInstanceDetails>
              </bindingTemplate>
            </bindingTemplates>
          </businessService>
        </businessServices>
      </businessEntity>
```

3.7.3.WADL

Web Application Description Language (kraće *WADL*) je strojno čitljiv *XML* opis *HTTP*-baziranih web aplikacija (tipično *REST* web servisa). *WADL* modelira resurse koje servis pruža i odnose među njima. Namijenjen je olakšavanju ponovne uporabe web servisa koji se temelje na postojećoj *HTTP* arhitekturi weba. Neovisna je o platformi i jeziku te ima cilj poticati ponovnu uporabu aplikacija izvan osnovne uporabe u web pregledniku [48].

3.7.3.1. Glavni elementi WADL-a

Poput svakog opisnog jezika, *WADL* ima svoje karakteristične elemente koji se moraju zadovoljiti prilikom specifikacije [49]:

- ***Resource*** (Resurs): Ovaj element omogućuje *WADL*-u da objasni resurse identificirane *URI*-jem. Sadrži osnovni atribut tipa *xsd:anyURI* i djeluje kao spremnik za sve resurse koje aplikacija nudi. Osnovni atribut ovog elementa pruža osnovni *URI* za svaki uključeni resurs djeteta. Ključni atributi ovog elementa uključuju *id*, *path*, *type*, *queryType*.
- ***Method*** (Metoda): Ovaj element WADL opisa odnosi se na ulaz i izlaz metode putem HTTP-a, implementiran za određeni resurs. Ključni elementi ovog odjeljka uključuju *doc*, *request* i *response*.
- **Reference metode**: To je podvrsta elementa *href* koji pripisuje resursnom elementu vrijednost tipa *xsd:anyURI*. Ovdje vrijednost atributa *href* djeluje kao križna referenca definicije elementa metode.
- **Definicija metode**: Dijete aplikacijskog ili resursnog elementa.
- ***Request*** (Zahtjev): Pomoću ovog elementa *WADL* opisuje ulaz koji se koristi za *HTTP* metodu. Ne sadrži atribute, ali odjeljak sadrži nekoliko elemenata poput *doc*, *representation* i *param*.

- ***Response*** (Odgovor): Dio različitih atributa koji definiraju *WADL*, odgovor označava izlaz primljen za poslan *HTTP* zahtjev. Zauzvrat, prima samo vrijednost atributa *status*. Status je neobavezan i pruža *HTTP* statusne kodove povezane s određenim odgovorom. Ključni elementi ovog odjeljka: uključuju *doc*, *representation* i *param*.
- **Parametar**: Parametar ili element *param* označava parametriziranu vrijednost roditeljskog elementa.
- **Referenca parametra**: Ovaj tip elementa *param* označava se atributom *href* s vrijednošću tipa *xsd:anyURI*. *href* je križna referenca za element definicije parametra. Standardna praksa je izuzeti atribute ili elemente djeteta definirane putem *WADL*-a iz nje, uzimajući u obzir njenu važnost. Najčešće se koristi za smanjenje dupliciranja u scenarijima u kojima se parametar primjenjuje na više roditelja.
- **Definicija parametra**: Ovaj tip definira parametriziranu vrijednost elemenata, kao što su roditeljski element ili resurs djeteta. Osim toga, definicija parametra koristi se i za elemente reprezentacije, aplikacije, odgovora i zahtjeva.

3.7.3.2. Primjer WADL specifikacije

U nastavku se nalazi konkretni primjer *WADL* specifikacije nad *RESTful* servisom, dizajniran za operacije nad skladištem, a konkretno radi nad resursima *containers* i *item* koristeći različite *HTTP* metode [50]:

```
<?xml version="1.0" encoding="UTF-8" skandalozna="yes"?>
<application xmlns="http://research.sun.com/wadl/2006/10">
    <doc xmlns:jersey="http://jersey.dev.java.net/"
            jersey:generatedBy="Jersey:   1.0-ea-SNAPSHOT   10/02/2008
12:17 PM"/>
    <resources base="http://localhost:9998/storage/">
        <resource path="/containers">
            <method name="GET" id="getContainers">
                <response>
                    <representation mediaType="application/xml"/>
                </response>
            </method>
            <resource path="{container}">
                <param xmlns:xs="http://www.w3.org/2001/XMLSchema"
                                                        type="xs:string"
style="template" name="container"/>
                <method name="PUT" id="putContainer">
                    <response>
                        <representation mediaType="application/xml"/>
                    </response>
```

```
                    </method>
                    <method name="DELETE" id="deleteContainer"/>
                    <method name="GET" id="getContainer">
                        <request>
                            <param
xmlns:xs="http://www.w3.org/2001/XMLSchema"

            type="xs:string" style="query" name="search"/>
                        </request>
                        <response>
                            <representation mediaType="application/xml"/>
                        </response>
                    </method>
                    <resource path="{item: .+}">
                        <param
xmlns:xs="http://www.w3.org/2001/XMLSchema"

      type="xs:string" style="template" name="item"/>
                        <method name="PUT" id="putItem">
                            <request>
                                <representation mediaType="*/*"/>
                            </request>
                            <response>
                                <representation mediaType="*/*"/>
                            </response>
                        </method>
                        <method name="DELETE" id="deleteItem"/>
                        <method name="GET" id="getItem">
                            <response>
                                <representation mediaType="*/*"/>
                            </response>
                        </method>
                    </resource>
                </resource>
            </resource>
        </resources>
    </application>
```

3.7.4.OpenAPI specifikacija

OpenAPI specifikacija definira način opisivanja *API-ja* temeljenih na *HTTP-u*, koji su obično *RESTful API-ji. OpenAPI* definicija dolazi u obliku *YAML* ili *JSON* datoteke koja opisuje ulaze i izlaze *API-ja*. Također može uključivati informacije poput mjesta na kojem je *API* smješten, koja je autorizacija potrebna za pristup, te druge detalje potrebne potrošačima i proizvođačima. Definicije mogu biti pisane ručno ili pomoću alata, ili čak generirane iz koda. Kada se *API* zapiše, kažemo da je opisan i tada postaje platforma koju alati i ljudi mogu koristiti. Tipičan način korištenja definicija *API-ja* je generiranje ljudima čitljive dokumentacije iz nje. *OpenAPI* definicija može se na prvi pogled činiti pomalo opsežnom, ali ćete u njoj pronaći iznimno korisne informacije [15].

3.7.4.1. Sintaksa OpenAPI

OpenAPI specifikacija , koja je od prije poznata i kao Swagger, upotrebljava *YAML Ain't Markup Language* (kraće YAML) ili *JSON* format za definiranje *RESTful* web *API-ja*. U nastavku su navedeni ključni dijelovi specifikacije [15]:

- ***openapi***: Definira verziju *OpenAPI* specifikacije koja se koristi.
- ***info***: Sekcija "info" sadrži osnovne informacije o *API-ju*. To uključuje naziv *API-ja*, opis koji pruža pregled njegove svrhe ili funkcionalnosti te verziju *API-ja*.
- ***servers***: Definira dostupne servere na kojima je *API* implementiran. Ovo može uključivati različite *URL-ove* ili domene na kojima je *API* dostupan, poput domena za produkciju ili testiranje.
- ***paths***: Svaka putanja predstavlja određeni *URL* do resursa ili funkcionalnosti *API-ja*.
- ***get, post, put, delete* itd.**: Metode koje određuju koje akcije su podržane za određene resurse.
- ***summary***: Kratak sažetak funkcionalnosti resursa.
- ***description***: Detaljan opis resursa.
- ***responses***: Definira moguće odgovore na zahtjev za određeni resurs. Svaki odgovor opisuje kako će *API* reagirati na određeni scenarij, uključujući *HTTP* statusni kod, moguće tijelo odgovora i druge detalje.
- ***example***: Primjer odgovora pruža konkretan uzorak podataka kako bi klijenti bolje razumjeli očekivanu strukturu i format odgovora za određeni resurs.

3.7.4.2. Primjer OpenAPI specifikacije

U nastavku se nalazi konkretni primjer OpenAPI specifikacije u YAML obliku koja opisuje jednostavan RESTful API za dohvaćanje informacija o knjigama:

```
openapi: 3.0.0
```

```
info:
  title: Book API
  description: RESTful API za dohvaćanje informacija o knjigama.
  version: 1.0.0
servers:
  - url: http://api.example.com
paths:
  /books:
    get:
      summary: Dohvati sve knjige
      description: Prikazuje popis svih dostupnih knjiga.
      responses:
        '200':
          description: Uspješan odgovor
          content:
            application/json:
              example:
                - id: 1
                  title: The Great Gatsby
                  author: F. Scott Fitzgerald
                - id: 2
                  title: To Kill a Mockingbird
                  author: Harper Lee
```

3.7.5. Standardi za semantičke servise

Semantički servisi su sastavni dio semantičkog weba jer servise čine strojno-čitljivima na detaljan i sofisticiran način. Ono što ih razlikuje od običnih servisa je sposobnost označavanja podataka na način koji omogućuje računalima detaljno razumijevanje podataka. Klasični web servisi se temelje na sintaksi, dok semantički web servisi dodatno donose semantičko značenje podataka, omogućujući automatiziranu kompoziciju web servisa. Problemi s klasičnim web servisima sežu od nedostatka semantičkog značenja u komunikaciji do teškoća u automatiziranoj kompoziciji servisa. Semantički web servisi rješavaju ove probleme oslanjajući se na univerzalne standarde razmjene semantičkih podataka, što omogućuje kombiniranje podataka iz različitih izvora i usluga bez gubitka značenja. Semantički web servisi unapređuju komunikaciju između web servisa, omogućuju automatizaciju i integraciju podataka te olakšavaju stvaranje složenih interakcija među uslugama i izvorima podataka [51].

3.7.5.1. SAWSDL

Semantic Annotations for WSDL and XML Schema (kraće *SAWSDL*) definira skup proširenih atributa koji omogućava opis dodatne semantike komponenata *WSDL*-a. Specifikacija definira kako se postiže semantička anotacija korištenjem referenci na semantičke modele, poput ontologija. *SAWSDL* ne specificira jezik za predstavljanje semantičkih modela. Umjesto toga, pruža mehanizme putem kojih se koncepti iz semantičkih modela, obično definirani izvan *WSDL* dokumenta, mogu referencirati unutar *WSDL* i *XML Schema* komponenata pomoću anotacija [52].

3.7.5.2. SA-REST

Semantic Annotations for REST (kraće *SA-REST*) je način dodavanja dodatnih meta-podataka u opise *REST API-ja* u *HTML* ili *XHTML* dokumentima. Ti meta-podaci mogu potjecati iz različitih modela poput ontologija ili taksonomija. Cilj je poboljšati pretraživanje, posredovanje podataka i integraciju usluga. *SA-REST* može poboljšati pretraživanje u direktorijima *API*-ja dodajući meta-podatke koji hvataju aspekte *API*-ja. *SA-REST* pomaže u lakšem posredovanju podataka, te mjeri ljudski trud za posredovanje podataka [53].

3.7.5.3. OWL

Web Ontology Language (kraće *OWL*) je jezik semantičkog weba osmišljen za prikazivanje bogatog i kompleksnog znanja o stvarima, grupama stvari i odnosima između stvari. *OWL* je jezik temeljen na računalnoj logici, tako da znanje izraženo u *OWL*-u može biti iskorišteno od strane računalnih programa, npr. za provjeru dosljednosti tog znanja ili za eksplicitno iznošenje implicitnog znanja. Dokumenti *OWL*-a, poznati kao ontologije, mogu biti objavljeni na globalnoj mreži i mogu se referirati na druge *OWL* ontologije ili biti referencirani iz drugih *OWL* ontologija. *OWL* je dio *W3C*-ovog skupa tehnologija semantičkog weba, koji uključuje *RDF*, *RDFS*, *SPARQL*, itd [54].

3.7.5.4. RDF

Resource Description Framework (kraće *RDF*) je standardni model kod razmjene podataka na webu. *RDF* ima značajke koje olakšavaju spajanje podataka čak i ako se temelja shema razlikuju, i posebno podržava evoluciju shema tijekom vremena bez potrebe za promjenom svih potrošača podataka. *RDF* proširuje strukturu povezivanja weba tako da koristi *URI*-jeve za imenovanje odnosa između stvari, kao i dvaju krajeva veza. Koristeći ovaj jednostavan model, omogućuje miješanje, izlaganje i dijeljenje strukturiranih i polustrukturiranih podataka putem različitih aplikacija. Ova struktura povezivanja oblikuje usmjereni označeni graf, gdje bridovi predstavljaju nazvane veze između dvaju resursa, predstavljenih čvorovima grafa [55].

4. Obrada prirodnog jezika

Obrada prirodnog jezika (eng. *Natural language processing - NLP*) je oblik umjetne inteligencije koji omogućuje računalima razumijevanje ljudskog jezika, bilo da je pisan, izgovoren ili čak nacrtan. Kako se uređaji i usluge pokretane umjetnom inteligencijom sve više isprepliću s našim svakodnevnim životom i svijetom, tako i utjecaj *NPL-a* postaje sve značajniji za osiguranje besprijekornog iskustva između ljudi i računala [8].

Organizacije danas posjeduju velike količine glasovnih i tekstualnih podataka iz različitih komunikacijskih kanala poput e-pošte, tekstualnih poruka, društvenih medija, videozapisa, zvuka i drugih. Koriste *NLP* softver za automatsku obradu tih podataka, analizu namjere ili sentimenta u poruci te odgovaranje u stvarnom vremenu na ljudsku komunikaciju [9].

4.1. Važnost obrade prirodnog jezika

Obrada prirodnog jezika ključna je za potpuno i učinkovito analiziranje tekstualnih i govornih podataka. Može raditi kroz razlike u dijalektima, slengu i gramatičkim nepravilnostima koje su uobičajene u svakodnevnim razgovorima. Tvrtke ga koriste za nekoliko automatiziranih zadataka, kao što su:

- Obrada, analiza i arhiviranje velikih dokumenata
- Analiza povratnih informacija korisnika ili snimaka iz call centara
- Pokretanje chatbotova za automatiziranu korisničku podršku
- Odgovaranje na pitanja tko-što-kada-gdje
- Klasifikacija i izdvajanje teksta

Također možete integrirati *NLP* u aplikacije koje su vidljive korisnicima kako biste komunicirali učinkovitije s klijentima. Primjerice, chatbot analizira i sortira upite korisnika, automatski odgovara na uobičajena pitanja i preusmjerava složene upite na podršku korisnicima. Ova automatizacija pomaže smanjenju troškova, štedi agentima vrijeme na redundantnim upitima i poboljšava zadovoljstvo korisnika [9].

4.2. Tehnike obrade prirodnog jezika

NLP obuhvaća širok raspon tehnika za analizu ljudskog jezika. Neke od najčešćih tehnika s kojima se stručnjaci u ovom području susreću su [10], [41]:

- **Analiza sentimenta**: Tehnika *NLP-a* koja analizira tekst kako bi prepoznala njegov sentiment, poput "pozitivnog", "negativnog" ili "neutralnog". Analiza sentimenta se često koristi u poslovanju kako bi se bolje razumjeli povratne informacije korisnika.
- **Sažimanje**: Tehnika koja sažima dulji tekst kako bi ga učinila upravljivijim za čitatelje koji nemaju puno vremena. Neki uobičajeni tekstovi koji se sažimaju uključuju izvješća i članke.
- **Ekstrakcija ključnih riječi**: Tehnika koja analizira tekst kako bi prepoznala najvažnije ključne riječi ili fraze. Ekstrakcija ključnih riječi se često koristi u optimizaciji pretraživača (eng. *Search engine optimization - SEO*), praćenju društvenih medija i svrhama poslovne inteligencije.
- **Tokenizacija**: Proces razbijanja znakova, riječi ili podriječi na "tokene" koji se mogu analizirati programom. Tokenizacija leži u osnovi uobičajenih zadataka *NLP-a* kao što su modeliranje riječi, izgradnja rječnika i učestala pojava riječi [8].
- **Modeliranje teme**: Tehnika koja koristi modele umjetne inteligencije za označavanje i grupiranje tekstualnih skupova na temelju ključnih riječi za temu i skupova informacija povezanih s njima.
- **Klasifikacija teksta**: Tehnika koja se bavi organiziranjem velikih količina sirovog teksta (koji se dobije od vlastitih korisnika). Modeliranje tema, analiza sentimenta i ekstrakcija ključnih riječi podskupovi su klasifikacije teksta. Klasifikacija teksta se često koristi za izvlačenje korisnih podataka iz recenzija korisnika kao i iz evidencija korisničke podrške.
- **Lematizacija**: Tehnika koja se koristi za smanjenje riječi na njihov osnovni oblik ili "lemu". Ovaj osnovni oblik riječi predstavlja korijenski oblik riječi i obično se odnosi na oblik koji se može naći u rječniku. Cilj lematizacije je dovesti različite oblike iste riječi na zajednički osnovni oblik, što olakšava analizu teksta. Na primjer, u lematizaciji se riječi "vozim", "voziš", "vozi" i "vozili" sve smanjuju na osnovni oblik "voziti". Na taj način, analiza teksta postaje konzistentnija i omogućava lakše prepoznavanje istog značenja u različitim varijacijama riječi.
- ***Stemming***: Tehnika koja se također koristi za smanjenje riječi na njihov osnovni oblik. Suprotno lematizaciji koja također uzima u obzir gramatička pravila i kontekst, *stemming* se često oslanja na jednostavne pravila za uklanjanje sufiksa ili prefiksa s

riječi kako bi se dobila njena osnovna forma. Primjerice, u stemmingu bi riječi poput "vozim", "voziš", "vozi" i "vozili" sve bile svedene na osnovni oblik "voz".

4.3. Neuronske mreže

Neuronska mreža je niz algoritama koji nastoje prepoznati temeljne odnose u skupu podataka putem procesa koji oponaša način rada ljudskog mozga. U tom smislu, neuronske mreže se odnose na sustave neurona, bilo organskih ili umjetnih. Neuronske mreže mogu se prilagoditi promjenjivim ulazima, tako da mreža generira najbolji mogući rezultat bez potrebe za redizajniranjem kriterija izlaza [56].

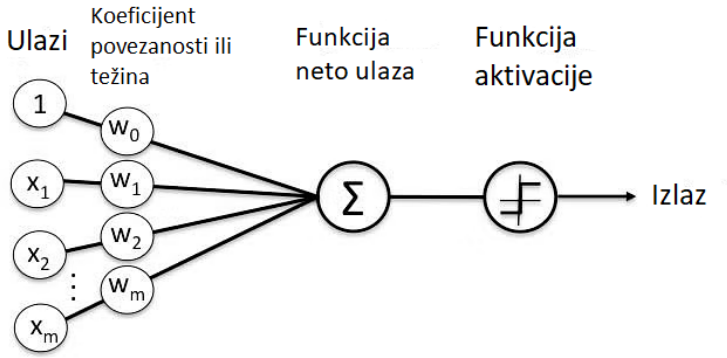
4.3.1. Tipovi neuronskih mreža

Nakon što smo stekli osnovno razumijevanje o neuronskim mrežama, sada ćemo istaknuti 6 najčešćih tipova neuronskih mreža [57]:

1. *Perceptron*
2. *Multi-layer Perceptron* (Više slojni perceptoni)
3. *Convolutional Neural Networks* (Konvolucijske neuronske mreže)
4. *Recurrent Neural Networks* (Rekurzivne neuronske mreže)
5. *Long Short Term Memory* Networks (Mreža dugoročne i kratkoročne memorije)
6. *Generative Adversarial Networks* (Generativne suparničke mreže)

4.3.1.1. Perceptron

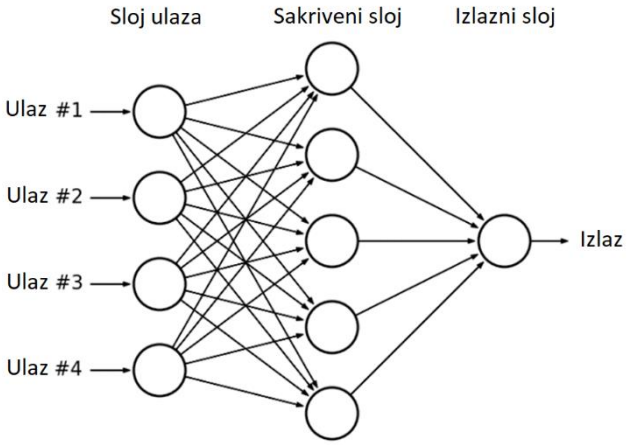
Perceptron je po strukturi najjednostavniji tip neuronske mreže. Model je također poznat kao jednoslojna neuronska mreža, koji sadrži samo dva sloja, a to su ulazni i izlazni sloj. Kod *Perceptrona* nema skrivenih slojeva. On preuzima ulazne podatke i izračunava ponderirani unos za svaki ulazni čvor. Taj ponderirani unos prolazi kroz funkciju aktivacije kako bi se generirao izlaz. Zbog jednostavne arhitekture, ne može se koristiti za složene zadatke. Umjesto toga, ova mreža se koristi za logičke izračune poput I (*AND*), ILI (*OR*) ili EKSKLUZIVNO ILI (*XOR*). *Perceptroni* se koriste u linearnoj ili binarnoj klasifikaciji modela. Također se koriste u formiranju višeslojnih perceptrona (*Multi-layer Percepton*) [57].



Slika 3: Perceptron model [63]

4.3.1.2. Više slojni perceptroni

Multi-layer Percepton ili kraće *MLP* predstavlja neuronske mreže s prosljeđivanjem unaprijed, obično označavaju potpuno povezane mreže. Drugim riječima, svaki neuron u jednom sloju je povezan sa svim neuronima u susjednim slojevima. Stoga *MLP* ima veću obradnu snagu od samostalnih perceptrona. *MLP* se koristi u različitim područjima. Uobičajena primjena prepoznata je kod kompresije podataka za društvene mreže, sustavima za prepoznavanje govora i rukom pisanih znakova, aplikacijama računalnog vida i sustavima za predviđanje podataka [57].

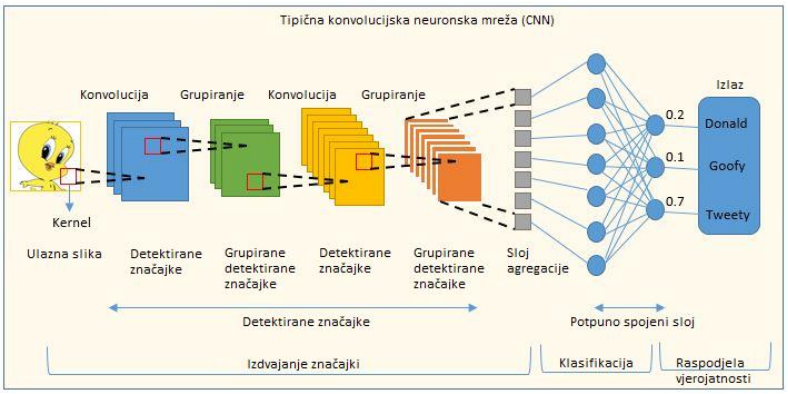


Slika 4: Model Više slojnih perceptrona [64]

4.3.1.3. Konvolucijske neuronske mreže

Ljudi prepoznaju objekte koristeći neurone u očima koji detektiraju rubove, oblike, dubinu i pokret. Jedan od najvažnijih tipova neuronskih mreža u računalnom vidu, *Convolutional Neural Networks* (kraće *CNN*) inspiriran je vizualnim korteksom očiju i koristi se

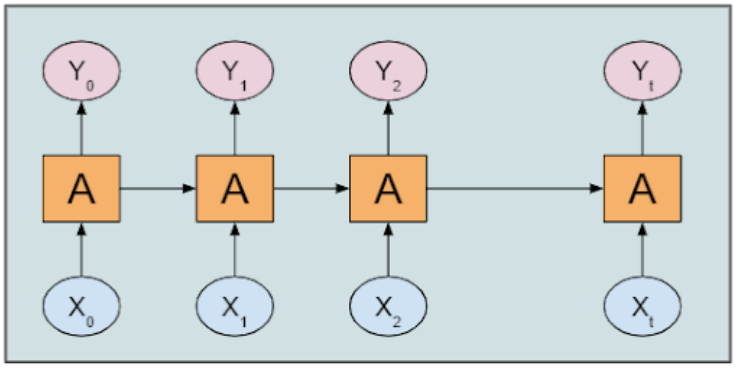
za vizualne zadatke poput detekcije objekata. Konvolucijski sloj *CNN*-a izdvaja ga od drugih neuronskih mreža. Taj sloj izvodi skalarni produkt, koji se sastoji od komponentne množenja, a zatim zbrajanja. Iako često zahtijevaju veliku količinu podataka za obuku, *CNN*-ovi se široko primjenjuju u različitim slikovnim i čak jezičnim zadacima. Budući da su *CNN*-ovi inspirirani vizualnim korteksom, široko se koriste za aplikacije koje uključuju primjenu računalnog vida. Te aplikacije uključuju prepoznavanje lica, detekciju lica, prepoznavanje objekata, prepoznavanje rukom pisanih slova i detekciju tumora u medicinskoj dijagnozi [57].



Slika 5: Model Konvoluncijskih neuronskih mreža [65]

4.3.1.4. Rekurzivne neuronske mreže

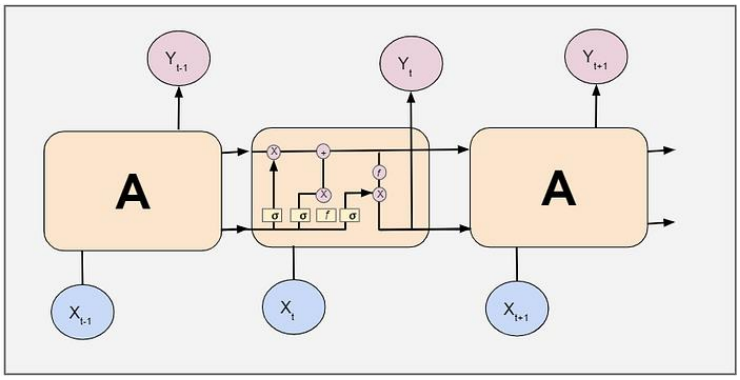
Knjiga se obično sastoji od niza poglavlja. Kada čitamo određeno poglavlje, ne pokušavamo ga razumjeti izolirano, već ga pokušavamo povezati s prethodnim poglavljima. Slično tome, kao i prirodne neuronske mreže, modeli strojnog učenja moraju razumjeti tekst koristeći već prethodno naučeni tekst. U tradicionalnim modelima strojnog učenja to je nemoguće jer ne možemo pohraniti prethodne faze modela. Međutim, *Recurrent Neural Networks* (kraće *RNN*) su vrsta neuronskih mreža koja to može učiniti za nas, čineći ih korisnima za aplikacije koje zahtijevaju upotrebu prošlih podataka. Rekurzivne neuronske mreže su mreže dizajnirane za tumačenje vremenskih ili sekvencijalnih informacija. *RNN*-ovi koriste druge točke podataka u sekvenci kako bi bolje predvidjeli. To čine tako što primaju ulaz i ponovno koriste aktivacije prethodnih čvorova ili kasnijih čvorova u sekvenci kako bi utjecali na izlaz. *RNN* ima ponavljajući modul koji preuzima ulaz iz prethodne faze i daje njegov izlaz kao ulaz sljedećoj fazi. *RNN*-ovi se često koriste u povezanim sekvencijskim aplikacijama poput prognoze vremenskih serija, obrade signala i prepoznavanja rukom pisanih znakova. Također, *RNN*-ovi se široko koriste u generiranju glazbe, titlovanju slika i predviđanju fluktuacija na burzama [57].



Slika 6: Model Rekurzivne neuronske mreže [57]

4.3.1.5. Mreža dugoročne kratkoročne memorije

U *RNN*-ovima možemo zadržati informacije samo iz najnovije faze. No, za problem kao što je prevođenje jezika, trebamo puno više zadržavanja. Tu nastupaju *Long Short-Term Memory Networks* (kraće *LSTM*). Za učenje dugoročnih ovisnosti, našoj neuronskoj mreži potrebna je sposobnost memoriziranja. *LSTM*-ovi su poseban slučaj *RNN*-ova koji to mogu postići. Imaju istu strukturu sličnu lancu kao *RNN*-ovi, ali s drugačijom strukturom ponavljajućeg modula. Ta struktura ponavljajućeg modula omogućuje mreži da zadrži puno veću količinu vrijednosti iz prethodnih faza. *LSTM*-ovi su moćni za sustave prevođenja jezika, ali imaju širok raspon primjena. Neke od tih primjena uključuju zadatke modeliranja sekvenci-prema-sekvenci kao što su otkrivanje anomalija, prepoznavanje govora, sažimanje teksta i klasifikacija videozapisa [57].

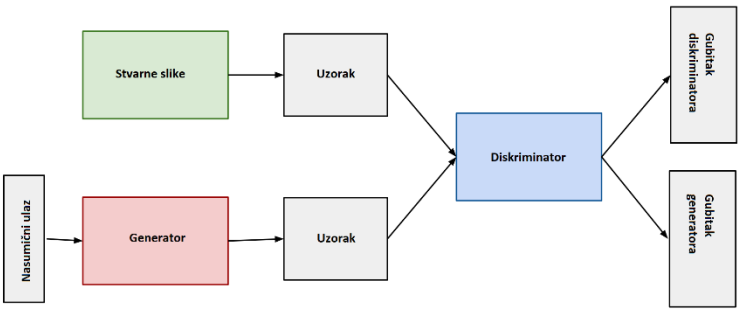


Slika 7: Model Mreža dugoročne kratkoročne memorije [57]

4.3.1.6. Generativne suparničke mreže

Dajući podatke za obučavanje, *Generative Adversarial Networks* **(**kraće *GAN*) uče generirati nove podatke s istom statistikom kao podaci za obučavanje. Na primjer, ako obučimo

GAN model na fotografijama, tada će obučeni model moći generirati nove fotografije koje izgledaju slično kao ulazne fotografije. *GAN* se sastoji od dvije komponente: generatora i diskriminatora. Model generatora stvara nove podatke dok diskriminator pokušava razlikovati stvarne podatke od generiranih podataka. Kako se generator i diskriminator sve bolje snalaze u svojim zadacima, generirani podaci se poboljšavaju, sve dok kvaliteta u idealnom slučaju nije gotovo identična podacima za obučavanje. Zamislite tu vezu kao onu između policije i pljačkaša. Oboje uvijek pokušavaju nadmudriti drugoga, pljačkaši da ukradu, a policija da uhvati pljačkaše. Koristeći generator, prvo stvaramo uzorke slučajnih šumova i provodimo ih kroz diskriminator. Diskriminator lako može razlikovati između ova dva tipa podataka, pa prilagođavamo model generatora i ponovno treniramo. S povećanjem iteracija, model generatora stvara podatke koji su neodvojivi od obučavajućih podataka. GAN-ovi se često koriste za stvaranje slika za crtane filmove, lica za igre i animirane filmove [57].



Slika 8: Model Generativne suparničke mreže [66]

4.4. Duboko učenje

Duboko učenje je grana strojnog učenja, metoda u području umjetne inteligencije. Model dubokog učenja sastoji se od nekoliko skrivenih slojeva neuronskih mreža koje izvode složene operacije na ogromnim količinama strukturiranih i nestrukturiranih podataka. Bavi se treniranjem računalnih sustava da obradu podataka vrše na način inspiriranim ljudskim mozgom [58], [59].

4.4.1. Primjena dubokog učenja

Kombinacijom neuronskih mreža, možemo konstruirati modele dubokog učenja koji imaju primjenu u različitim kontekstima [59]:

- **Financijske prognoze**: U financijskoj industriji, duboko učenje se koristi za predviđanja cijena dionica, valuta i slično. Aplikacije koriste prošlu izvedbu dionica, omjere neprofitabilnosti i godišnje prinose kao ulaz kako bi pružile predviđanja koja pomažu financijskim izvršiteljima da u stvarnom vremenu donose tržišna predviđanja.
- **Samostalna vozila**: Upotrebljavajući podatke prikupljene u stvarnom vremenu nad više od milijun vozila, automobilska kompanija *Tesla* koristi neuronske mreže kako bi se autonomna vozila mogla upravljati nailaskom na semafore i pri vožnji složenim mrežama ulica, kako bi se pronašla optimalna ruta. Koriste podatke iz senzora, kamera i radara kako bi otkrili ljude, objekte i druga vozila u okolini vozila.
- **Analiza korisničkog ponašanja**: Neuronske mreže obrađuju i analiziraju velike količine sadržaja koje korisnici generiraju na društvenim medijima, web stranicama i mobilnim aplikacijama. Cijenjeni uvidi izvedeni iz ovog procesa koriste se za kreiranje ciljanih marketinških kampanja temeljenih na stavovima korisnika, aktivnostima i povijesti kupnje.
- **Identifikacija oboljenja**: Neuronske mreže se sve više koriste u zdravstvu za otkrivanje ozbiljnih bolesti poput raka, upravljanje kroničnim bolestima i otkrivanje nepravilnosti u medicinskim slikama.
- **Kriminalistički nadzor**: Iako je njegova upotreba kontroverzna, neki dužnosnici pravosudnih tijela koriste duboko učenje za otkrivanje i sprječavanje zločina. U tim slučajevima, konvolucijske neuronske mreže koriste algoritme za prepoznavanje lica kako bi usporedile ljudska lica s ogromnom količinom digitalnih slika kako bi otkrile neobično ponašanje, slale upozorenja o sumnjivim aktivnostima ili identificirale poznate bjegunce.

4.5. Primjena NLP u stvarnom svijetu

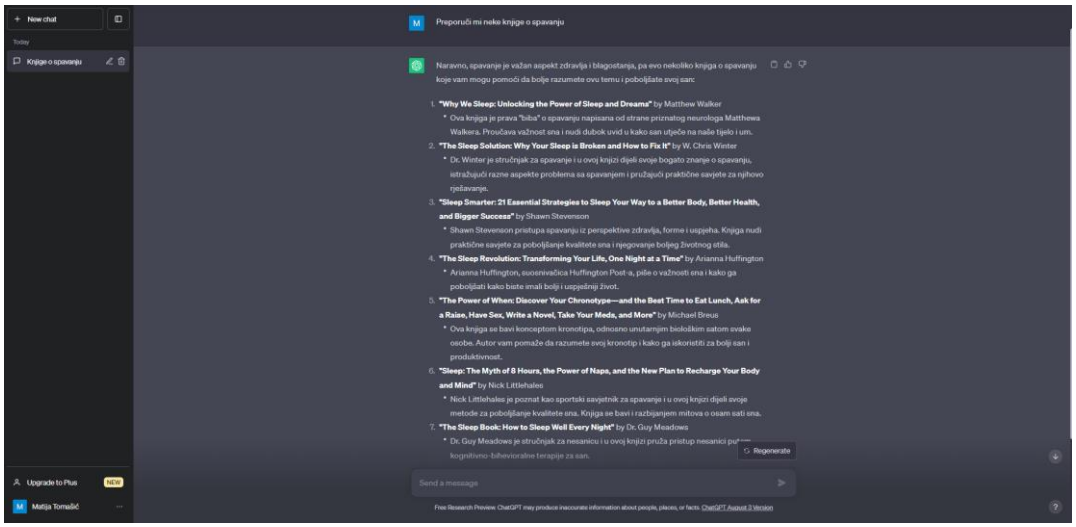
Obrada prirodnog jezika ima širok spektar primjena u stvarnom svijetu. Evo nekoliko primjera kako se *NLP* koristi u različitim industrijama [11]:

- **Rezultati tražilice**: Tražilice koriste obradu prirodnog jezika kako bi predložile relevantne rezultate na temelju prethodnog ponašanja u povijesti pretraživanja i namjere korisnika. Kada koristite *Google*, na primjer, tražilica predviđa što ćete nastaviti tipkati na temelju popularnih pretraga, istovremeno razmatrajući kontekst i prepoznajući značenje iza onoga što želite reći (za razliku od doslovno tipkanih riječi). Može se činiti kao da vaša misao biva dovršena prije nego što imate priliku završiti tipkanje.

- **Chatbotovi**: Chatbotovi su primjer primjene obrade prirodnog jezika u korisničkoj podršci. Korisni su za vlasnike Internet trgovina jer omogućuju korisnicima brze i na zahtjev odgovore na njihove upite. To je važno, posebno za manje tvrtke koje nemaju resurse za angažiranje stalnog agenta koji će pružati podršku korisnicima.
- **Automatsko prevođenje i automatsko prepoznavanje jezika**: Današnje aplikacije za prevođenje koriste obradu prirodnog jezika i strojno učenje kako bi točno prevele tekst i glasovne formate na većinu svjetskih jezika.
- **Automatsko ispravljanje pogrešaka u pisanju**: Današnje aplikacije za prevođenje koriste obradu prirodnog jezika i strojno učenje kako bi točno prevele tekst i glasovne formate na većinu svjetskih jezika.
- **Filtriranje mailova i prepoznavanje *spam-a***: Filtriranje e-pošte uobičajeni je primjer obrade prirodnog jezika koji možete pronaći na većini poslužitelja. Filtar za *spam* je gdje je sve počelo - otkrili su uzorke riječi ili fraza koje su bile povezane s spam porukama.
- **Pametno predlaganje proizvoda**: Preporuke proizvoda obično su temeljene na ključnim riječima. Ono što upišete je ono što ćete dobiti kao rezultat. S druge strane, obrada prirodnog jezika može uzeti u obzir više čimbenika, poput prethodnih podataka o pretraživanju i konteksta. Ti čimbenici mogu pomoći da rezultati pretraživanja budu precizniji. Također pomaže trgovcima da zadrže zanimanje posjetitelja preporukom pravih stvari. Ako prikažete proizvode koji odgovaraju potrebama kupaca, smanjit će se napuštanje web stranice i povećati broj kupnji [12].

4.5.1. ChatGPT

ChatGPT je *AI* jezični model razvijen od strane *OpenAI-ja* koji koristi duboko učenje kako bi generirao tekst sličan ljudskom. Koristi transformer arhitekturu, vrstu neuronske mreže koja je bila uspješna u različitim zadacima obrade prirodnog jezika i trenira se na masivnoj količini tekstualnih podataka kako bi generirao prirodni jezik u kontekstu interakcije s korisnikom. Cilj *ChatGPT*-a je generirati jezik koji je koherentan, kontekstualno primjeren i zvuči prirodno. *ChatGPT* se temelji na nekoliko najsuvremenijih tehnologija, uključujući obradu prirodnog jezika, strojno učenje i duboko učenje. Ove tehnologije se koriste za stvaranje dubokih neuronskih mreža modela i omogućuju mu učenje iz tekstualnih podataka i generiranje teksta [13].



Slika 9: Pregled ChatGPT na stolnom računalu

4.5.1.1. Pregled svih GPT modela kroz povijest

U nastavku će biti detaljno analizirane različite verzije *GPT* modela, s naglaskom na poboljšanja i dodatne uvedene u svakoj sljedećoj verziji [60]:

1. ***GPT-1***: Prvi je model u seriji *GPT* modela i treniran je na oko 40 GB tekstualnih podataka. Model je postigao rezultate koji su bili najbolji u modeliranju zadataka kao što su *LAMBADA* te je pokazao konkurentske performanse u zadacima poput *GLUE* i *SQuAD*. S maksimalnom duljinom konteksta od 512 tokena (oko 380 riječi), model je mogao zadržati informacije za relativno kratke rečenice ili dokumente po zahtjevu. Impresivne sposobnosti generiranja teksta i snažne performanse na standardnim zadacima potaknule su razvoj sljedećeg modela u seriji.
2. ***GPT-2***: Proizlazi iz modela *GPT-1*, zadržava iste arhitekturne značajke. Međutim, prošao je kroz treniranje na još većem opsegu tekstualnih podataka u usporedbi s *GPT-1*. Primjetno je da *GPT-2* može primiti dvostruko veći unos, omogućavajući mu obradu opsežnijih uzoraka teksta. S gotovo 1,5 milijardi parametara, *GPT-2* bilježi značajan skok u kapacitetu i potencijalu za modeliranje jezika. *GPT-2* koristi naprednije algoritme uzorkovanja u usporedbi s *GPT-1*. Prilikom konstrukcije *GPT-2* dolazi do podešavanje temperature logita, što dovodi do kontrole razine slučajnosti u generiranom tekstu. Niže temperature proizvode konzervativniji i predvidljivi tekst, dok više temperature stvaraju kreativniji i neočekivani tekst.
3. ***GPT-3***: Ovaj model predstavlja evoluciju *GPT-2* modela te ga nadmašuje u nekoliko ključnih aspekata. Prošao je obuku na značajno opsežnijem korpusu tekstualnih podataka i posjeduje maksimalno 175 milijardi parametara. U usporedbi s prethodnim modelima, *GPT-3* model donosi inovaciju omogućavajući generiranje teksta kao

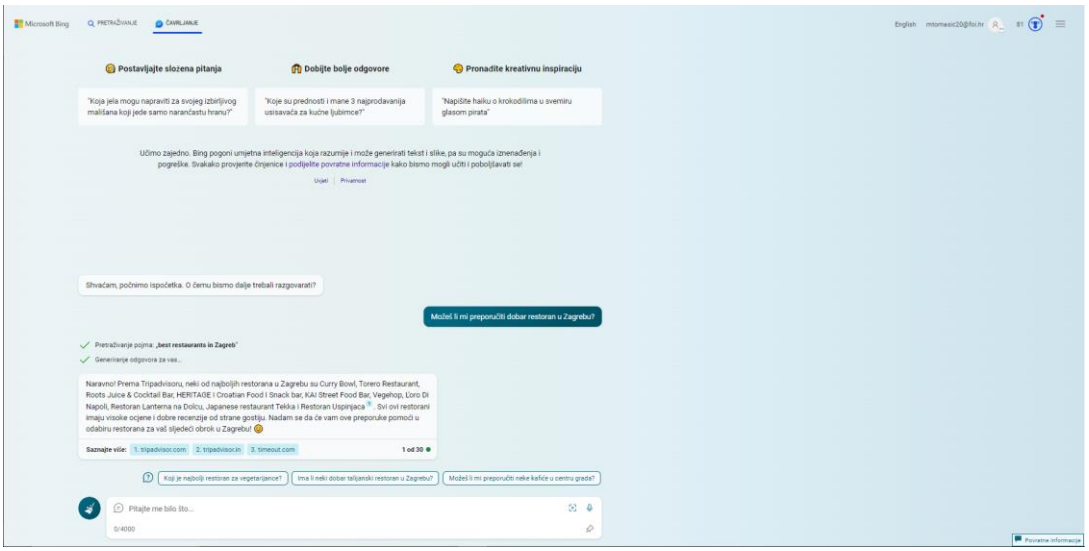
odgovor na potpuno nove zahtjeve, oslanjajući se na svoje općenito razumijevanje jezika i zadatka. Njegova sposobnost učenja s malim brojem primjera omogućuje brzu prilagodljivost novim zadacima i domenama s minimalnim potrebama za treniranjem. Model ističe impresivnu sposobnost brzog učenja na temelju ograničenog broja primjera. Bitno je naglasiti da *GPT-3* donosi i višejezičnu podršku, generirajući tekst na oko 30 jezika, uključujući engleski, kineski, francuski, njemački i arapski. Algoritam uzorkovanja koji *GPT-3* koristi je poboljšan, omogućujući prilagodbu razine slučajnosti u generiranom tekstu, slično kao i kod *GPT-2* modela.

4. ***GPT-3.5***: Model koji je kod obuke koristio preko 570 GB podataka. *GPT-3.5* serija modela, temelje se na prethodnim postignućima modela *GPT-3*, ali donosi karakterističnu inovaciju. Ta inovacija leži u integraciji strogo definiranih smjernica utemeljenih na ljudskim vrijednostima. To je postignuto primjenom tehnike pod nazivom *Reinforcement Learning with Human Feedback* (Povećanje učenja uz povratnu informaciju od ljudi) ili kraće *RLHF*. Osnovni cilj ove tehnike bio je približiti modele korisničkim namjerama, smanjiti prisutnost toksičnosti te postaviti istinitost kao prioritet u generiranom izlazu. Ova evolucija označava svjesne napore za unapređenje etičke i odgovorne uporabe jezičnih modela, sa svrhom osiguranja sigurnijeg i pouzdanijeg korisničkog iskustva. *OpenAI* je primijenio *RLHF* tehniku za precizno podešavanje *GPT-3* modela i osiguranje usklađenosti sa širokim spektrom uputa. Tehnika *RLHF* uključuje treniranje modela kroz načela povećanja učenja, pri čemu model dobiva nagrade ili kazne na temelju kvalitete i usklađenosti njegovih generiranih izlaza s procjeniteljima ljudske kvalitete. Integracija ovog povratnog utjecaja u proces obuke omogućuje modelu da uči iz svojih grešaka i kontinuirano poboljšava svoje performanse, rezultirajući krajnjim izlazima koji su prirodniji i angažirajući za korisnike.
5. **GPT-4**: Najnoviji model u seriji *GPT* modela i koji uvodi multimodalne sposobnosti koje omogućavaju obradu tekstualnih i slikovnih unosa dok generira tekstualne izlaze. Može obraditi različite formate slika, uključujući dokumente s tekstom, fotografije, dijagrame, grafikone, sheme i snimke zaslona. Iako *OpenAI* nije otkrio tehničke detalje kao što su veličina modela, arhitektura, metodologija treniranja ili težine modela za *GPT-4*, neki procjenjuju da ima gotovo 1 milijardu parametara. Osnovni model *GPT-4* slijedi cilj treniranja sličan prethodnim *GPT* modelima, teži predviđanju sljedeće riječi na temelju niza riječi. Proces treniranja uključivao je korištenje ogromne količine javno dostupnih internetskih podataka i licenciranih podataka. *GPT-4* je pokazao superiornu izvedbu u usporedbi s *GPT-3.5* u internim evaluacijama faktualnosti OpenAI-a i javnim mjerilima poput *TruthfulQA*. Tehnike *RLHF* koje su se koristile u *GPT-3.5* također su ugrađene u *GPT-4*. *OpenAI* aktivno nastoji unaprijediti *GPT-4* na temelju povratnih informacija dobivenih od *ChatGPT* i drugih izvora.

4.5.1.2. Bing Chat

Bing Chat predstavlja alternativu *ChatGPT*-u i proizvod je kompanije *Microsoft*, temeljen na najnovijem modelu *GPT-4*. Pristup mu je omogućen putem *Microsoft Edge* web preglednika. Dok i *ChatGPT* i *Microsoftov Bing Chat* funkcioniraju slično, postoji nekoliko značajnih razlika u njihovom pristupu i funkcionalnostima [61]:

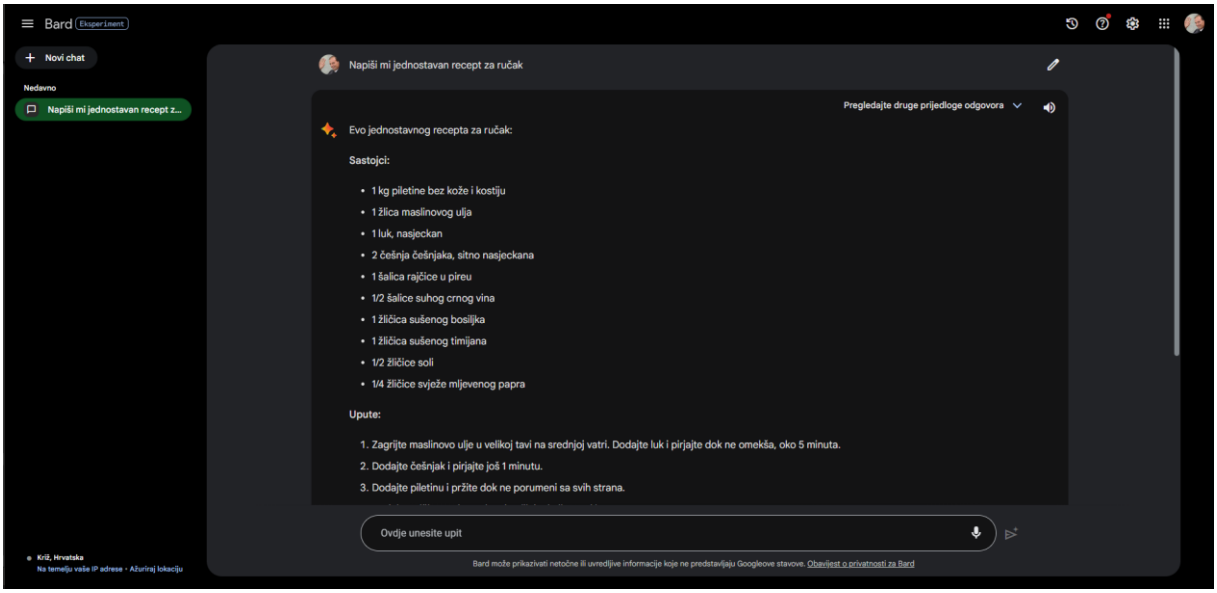
- **Sučelje i stil konverzacije**: Iako oba modela koriste sličan pristup unosa upita, *ChatGPT* i *Bing Chat* imaju različit format odgovora, konverzacijski stil i sučelje. *Bing Chat* pruža odgovore u različitim formatima, uključujući kreativne, uravnotežene i precizne odgovore, kako bi bolje zadovoljio potrebe korisnika.
- **Dodavanje slike**: *Bing Chat* omogućuje korisnicima da dodaju slike za obradu putem *AI*, što je funkcionalnost slična *Google Lensu*. Ovo je značajna razlika u odnosu na *ChatGPT*, koji se fokusira isključivo na tekstualne upite.
- **Mikrofon**: *Bing Chat* korisnicima omogućuje da koriste mikrofon svog uređaja umjesto tipkovnice za davanje uputa *AI* chatbotu. Ovo je praktična značajka koja omogućuje razgovor s *AI* koristeći govor umjesto teksta.
- **Izvori i poveznice**: Kada *Bing Chat* pruži odgovor na upit, on također može navedene izvore i poveznice ispod odgovora. Ovo pomaže korisnicima da bolje razumiju informacije i provjere izvore.
- **Prebacivanje između pretraživanja i razgovora**: *Bing Chat* omogućuje korisnicima da prelaze između tradicionalnih rezultata pretraživanja i *AI* chatbota. To znači da možete odabrati između klasičnog pretraživanja i interakcije s chatbotom.
- **Ograničenje odgovora po konverzaciji**: Bing Chat ima ograničenje od 30 odgovora po konverzaciji. To znači da će razgovor biti prekinut nakon što se dostigne taj broj odgovora.



Slika 10: Pregled Bing Chat-a na stolnom računalu

4.5.2.Bard AI

Još jedan konkurentan sustav u svijetu chatbotova u 2023. godini, dolazi iz kompanije Google, a naziva se *Bard AI*. *Bard AI* je temeljen na *PaLM 2* jezičnom modelu, treniran na masivnom skupu podataka teksta i trenutno je još u eksperimentalnom fazi razvoja, ali ima potencijal postati vrijedan alat za komunikaciju. U stanju je generirati različite kreativne tekstualne formate poput pjesama, koda, skripti, glazbenih djela i slično [62].



Slika 11: Pregled Bard AI na stolnom računalu

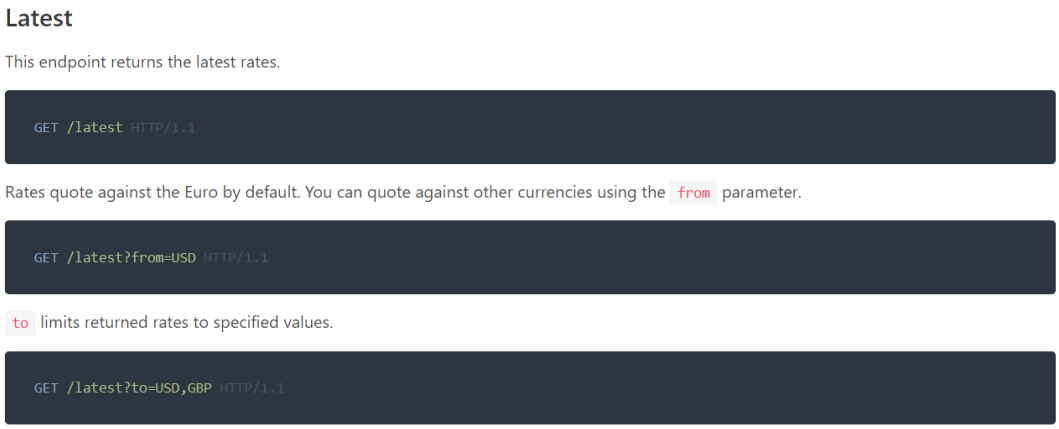
5. Implementacija aplikacije za generiranje opisa API-ja

U ovom poglavlju će temeljito biti opisan proces stvaranja aplikacije koja ima ključnu ulogu u automatiziranom stvaranju detaljnog opisa za jednostavan *RESTful* servis. Osim što će biti objašnjeni tehnički koraci potrebni za razvoj ove aplikacije, biti će pojašnjen njen značaj u kontekstu olakšavanja interakcija između web servisa i klijenata.

Nadalje, biti će razmotreni primjeri *API-ja* s i bez strojno čitljivih opisa te će biti ilustrirani kako strojno razumljiva specifikacija, poput *OpenAPI* formata, znatno unapređuje komunikaciju između različitih dijelova sustava. Uz to, biti će predočen čitljiv opis generiran od strane naše aplikacije, oblikovan u *OpenAPI* specifikaciji, kako bi se jasno prikazala korist koju ovakva automatizacija pruža. Na kraju će se demonstrirati testiranje generirane specifikacije.

5.1. Pregled API-ja bez strojno čitljivog opisa

Kao ilustraciju *API-ja* bez strojno čitljivog opisa, razmotrimo *The Frankfurter API*. *The Frankfurter API* prati referentne tečajeve stranih valuta objavljene od strane Europske središnje banke. Podaci se osvježavaju oko 16:00 CET svakog radnog dana [16]. Na njihovoj web stranici, u nestandardiziranom obliku, vidljiv je popis resursa i njihov sažeti opis, koji bi trebao pružiti informacije o podacima koje ti resursi vraćaju.



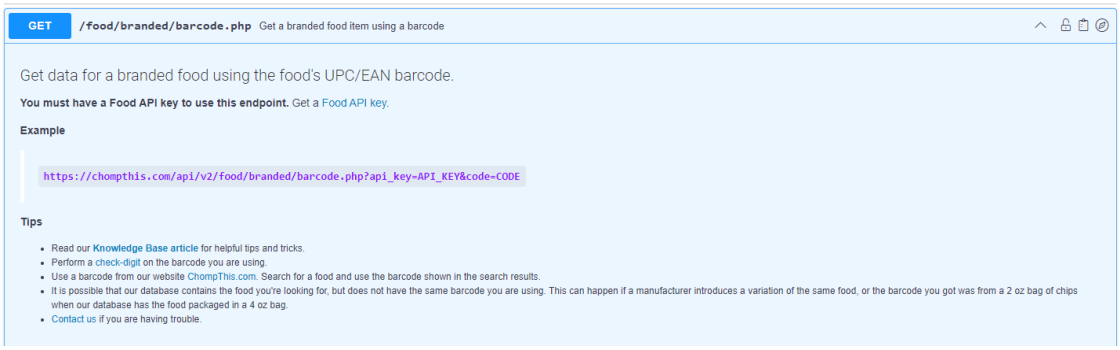
Slika 12: Pregled Frankfurter API-ja na stolnom računalu

5.2. Pregled API-ja opisanog OpenAPI specifikacijom

Kao primjer *API-ja* sa strojno čitljivim opisom, razmotrimo *Chomp Food & Recipe Database API.* Navedeni *API* pohranjuje podatke o više od 875 000 prehrambenih proizvoda, namirnica i sirovih sastojaka u Sjedinjenim Američkim državama i diljem svijeta [17]. Na zasebnoj stranici koja je rezervirana za dokumentaciju, dostupna je detaljna *OpenAPI* specifikacija koja pruža duboke opise korištenja *API-ja*. U ovoj dokumentaciji temeljito su opisani resursi, povratni podaci, obavezni parametri unosa, korisni savjeti i niz ilustrativnih primjera pogrešaka koji pokrivaju raznolike scenarije upotrebe. Na slici 4. vidljiv je pregled svih dostupnih resursa kojima je moguće pristupiti preko *API-ja*.

Slika 13: Pregled resursa Chomp Food & Recipe API-ja na stolnom računalu

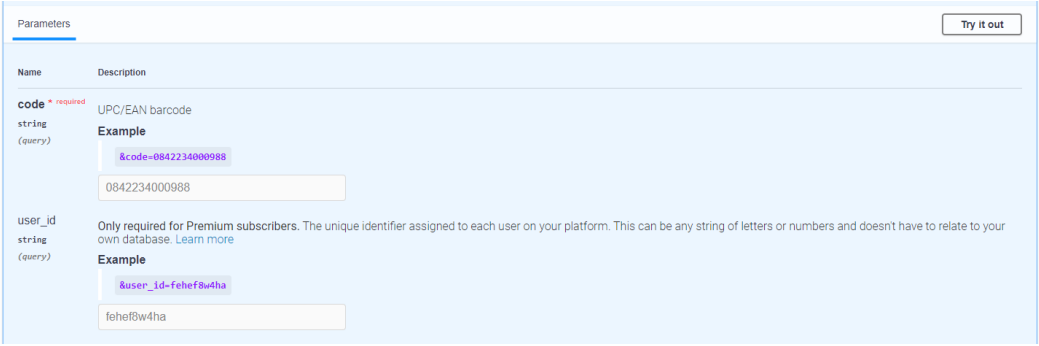
Na slici 5. specifično je vidljiv primjer resursa */food/branded/barcode.php* koji dohvaća *brendiranu* hranu pomoću barkoda. Cjelokupna putanja je jasno vidljiva, zajedno s korisnim uputama za pravilnu uporabu tog resurs



Slika 14: Pregled resursa koji se bavi dohvaćanjem hrane prikazan na stolnom računalu

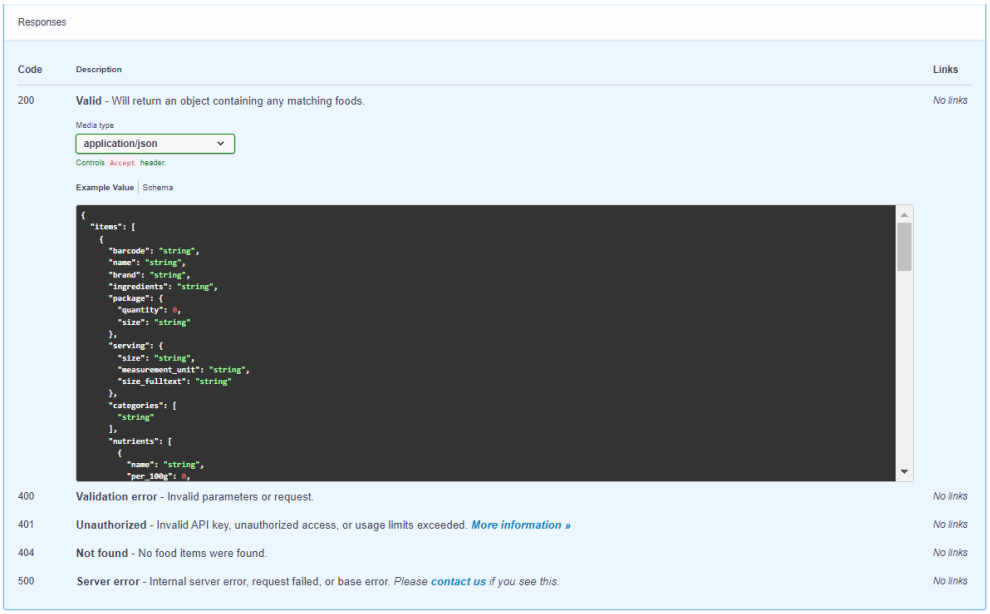
Na slici 6. vidljiva su rezervirana polja za unos parametara, gdje je moguće testirati *API* stvarnim podacima koja su pohranjeni u bazi. Konkretno za resurs */food/branded/barcode.php* ključna su tri parametra:

- *code:* Barkod proizvoda kojeg želimo dohvatiti. To nam omogućuje ciljano traženje i pristup informacijama o određenom proizvodu.
- *user_id*: Jedinstveni identifikator registriranog *premium* korisnika. Koristeći *user_id*, sustav može prilagoditi iskustvo korisnika prema njihovim individualnim postavkama i preferencijama.
- *api_key*: Ključ za autorizaciju, *api_key* osigurava ispravan pristup *API-ju*. Ovaj ključ se unosi na samom početku dokumentacije kako bi se osigurala sigurnost i kontrolirani pristup.



Slika 15: Prikaz polja za unos parametara odabranog resursa na stolnom računalu

Na slici 7. možemo vidjeti konkretne primjere odgovora koje klijent može očekivati nakon što je poslao zahtjev.



Slika 16: Prikaz primjera odgovora na pozivanje zadanog resursa na stolnom računalu

5.3. Početak implementacije aplikacije

Svrha ove web aplikacije je omogućiti klijentima da pristupe dokumentaciji *RESTful API-ja* na jedinstven i standardiziran način. Iako možda nije globalno jednako prepoznat, *OpenAPI* specifikacija predstavlja široko prihvaćen standard za jasno prikazivanje kako operativnih karakteristika, tako i nefunkcionalnih aspekata nekog *API* sučelja. Naša web aplikacija je koncipirana tako da podržava korisnike na računalima i mobilnim uređajima, dok se proces generiranja specifikacije odvija centralizirano na poslužitelju. U nastavku će biti opisan postupak razvoja ove aplikacije demonstrirajući svaki korak koji je doveo do stvaranja funkcionalnog i korisnički orijentiranog proizvoda.

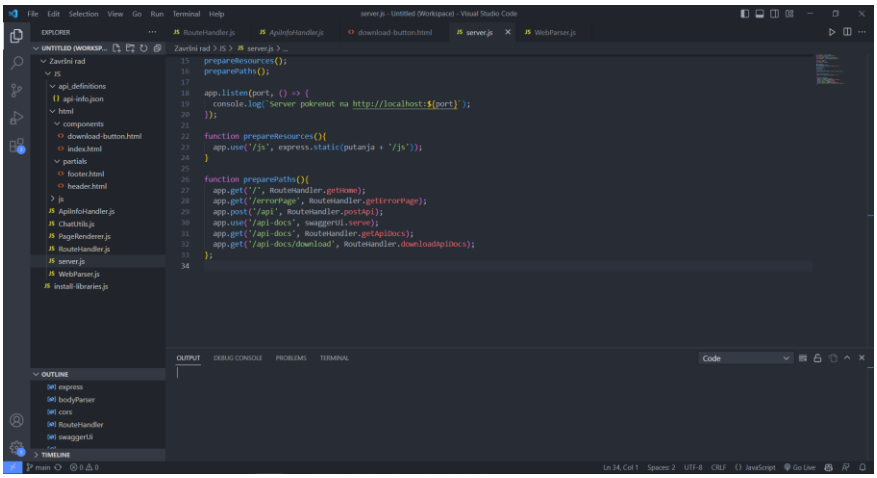
5.3.1. Postavljanje razvojnog okruženja

Ovisno o potrebama projekta i funkcionalnostima koje želimo implementirati, pažljivo biramo odgovarajuće alate i tehnologije za razvoj kako bismo osigurali učinkovit i produktivan proces izrade. Imajući na umu da naša web aplikaciju neće biti kompleksna i neće za to biti potrebno razvijanje u složenom okruženju, fokusiramo se na odabir alata koji su intuitivni i jednostavni za korištenje. Prilikom planiranja, nije bilo nužno razmatrati skalabilnost aplikacije

jer naš cilj nije izgradnja visoko skalabilne platforme. Umjesto toga, fokus je na konkretnom zadatku: demonstracija potencijala standardizacije *RESTful* web servisa i iskorištavanje *GPT* modela za automatizaciju generiranja opisa web servisa na brz i jednostavan način.

5.3.1.1. Odabira alata i tehnologija za razvoj

Kao alat za razvoj koda, odabran je *Visual Studio Code* - moćan alat s intuitivnim sučeljem koji omogućava efikasno pisanje i uređivanje koda. Za poslužiteljsku komponentu, korišten je *Node.js*, postavljajući ga kao lokalni poslužitelj unutar mreže koje radimo. Sama aplikacija razvijena je u *JavaScriptu*. Sučelje aplikacije sastavljeno je uz pomoć razvojnog okvira *Bootstrap* koji pojednostavljuje pisanje *HTML*, *CSS* i *JavaScript* dijela koda.



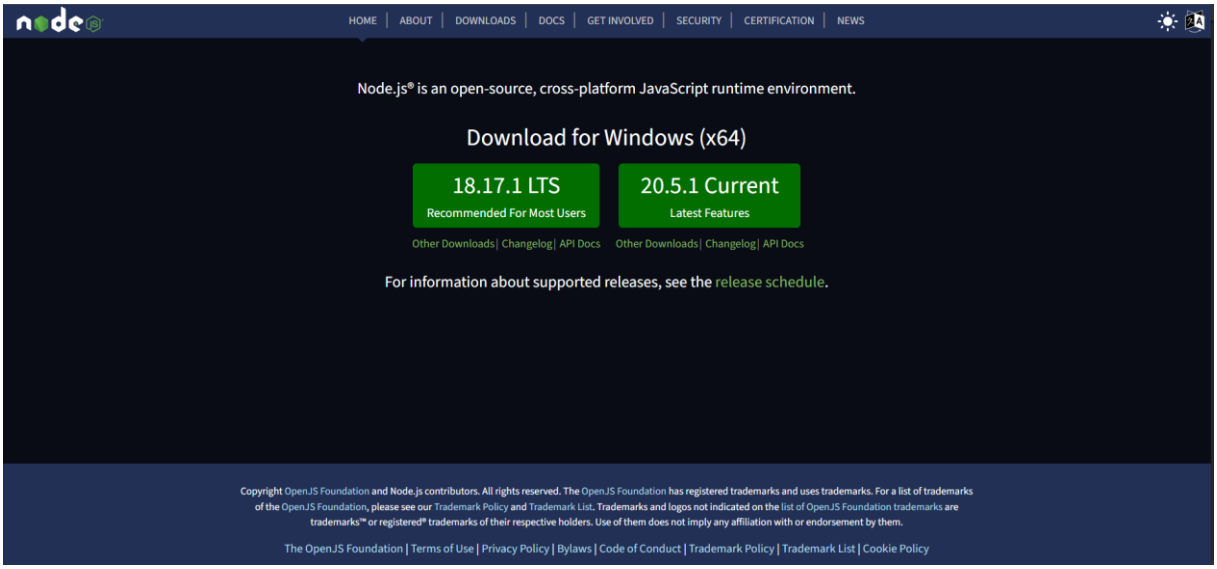
Slika 17: Sučelje Visual Studio Code-a [autorski rad]

5.3.1.2. Instalacija Node.js okruženja

Node.js je moćno razvojno okruženje koje omogućava izgradnju dinamičnih mrežnih aplikacija koristeći *JavaScript* programski jezik. Neovisno o tome koristite li *Windows*, *macOS* ili *Linux* operativni sustav, *Node.js* vam pruža svestranost da razvijate aplikacije koje će brzo i učinkovito komunicirati s poslužiteljem [40]. U kontekstu našeg projekta, *Node.js* ima ključnu ulogu u podršci za razvoj back end komponente aplikacije. Koristimo ga kako bismo osigurali da naša aplikacija bude dostupna klijentima te kako bismo implementirali pažljivo osmišljene funkcionalnosti.

Slika 18: Logo Node.js-a [33]

Postupak preuzimanja Node.js-a s njegove službene stranice je jednostavan i pristupačan. Nakon preuzimanja, slijedi se dokumentirana instalacijska procedura koja je prilagođena operativnom sustavu za koju se web aplikacija razvija.



Slika 19: Pregled Node.js početne stranice na stolnom računalu

5.3.1.3. Instalacija i postavljanje pripadajućih Node.js biblioteki i razvojnih okvira

Za implementaciju planiranih funkcionalnosti, ključno je instalirati potrebne biblioteke u *Node.js* okruženju. Sve te biblioteke mogu se lako instalirati putem naredbe *npm*, koja je dio šireg sustava poznatog kao upravitelj paketa za *Node.js*. Kreiran je 2009. godine kao *open-source* projekt kako bi pomogao *JavaScript* programerima da lako dijele pakirane module koda [18].

Specifično za ovaj projekt potrebno je instalirati sljedeće *Node.js* biblioteke na globalnoj razini, kako bi inicijalno pokretanje aplikacije prošlo uspješno:

- ***gpt-3-encoder***: Biblioteka koja služi za enkodiranje teksta kako bi se mogao lakše obraditi i koristiti u komunikaciji s *GPT* modelom. U našem slučaju, važno je prebrojati broj tokena koji se šalje u komunikaciji s *GPT* modelom.
- ***openai***: Biblioteka koja služi za slanje zahtjeva i primanje odgovora *GPT* modela.
- ***normalize-space***: Biblioteka koja služi za normalizaciju praznina i razmaka u tekstualnim nizovima, konkretno da se minimalizira broj tokena koji se šalje kao zahtjev prema *GPT* modelu.
- ***swagger-ui-express***: Biblioteka koja pruža generiranje interaktivnog sučelja temeljenog na proslijeđenoj *OpenAPI* specifikaciji.
- ***cors***: Biblioteka koja omogućava i rješava problem komunikacije između različitih domena.
- ***axios***: Biblioteka koja olakšava slanje *HTTP* zahtjeva prema poslužitelju iz kontroliranog *Node.js* okruženja.
- ***jsdom***: Služi za manipulaciju *Document Object Model-a* (kraće DOM), konkretno u našem slučaju služi za dohvaćanje samo tekstualnog dijela dokumentacije, budući da više od toga ne treba.

Kako bi olakšali proces instalacije potrebnih biblioteki, spremna je JavaScript skripta koja u jednom pokretanju, instalira sve potrebne biblioteke na globalnoj razini. Ona se pokreće naredbom *node install-libraries.js*.

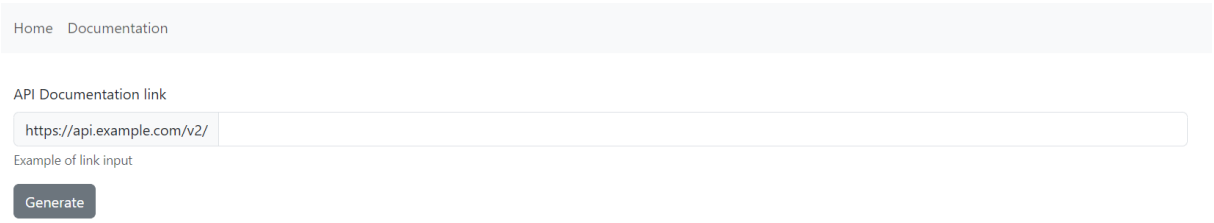
```
const { execSync } = require('child_process');
const librariesToInstall = [
  'gpt-3-encoder',
  'openai',
  'normalize-space',
  'swagger-ui-express',
  'cors',
  'axios',
  'jsdom'
];

librariesToInstall.forEach(library => {
  console.log(`Installing ${library}...`);
  try {
    execSync(`npm install -g ${library}`, { stdio: 'inherit' });
    console.log(`${library} installed successfully.`);
```

```
  } catch (error) {
    console.error(`Failed to install ${library}.`);
  }
});
```

5.3.2. Početna stranica

Na početnoj stranici web aplikacije, nalazi se forma za unos putanje do dokumentacije koju želimo analizirati. Uz polje za unos, nalazi se primjer kako bi valjana putanja trebala biti strukturirana. U slučaju da se unese nevažeća putanja, aplikacija će vratiti da se dogodila pogreška.



Slika 20: Prikaz forme za unos putanje do dokumentacije [autorski rad]

5.3.3. Dohvaćanje dokumentacije

Nakon što se unese putanja do dokumentacije, slijedi proces dohvaćanja sadržaja sa te stranice, što se uz pomoć prethodno spomenute biblioteke *axios* izvodi na sljedeći način:

```
async fetchResponse(file){
      return await axios.get(file);
}
```

5.3.4. Parsiranje dokumentacije

Nakon što se dohvati dokumentacija, potrebno je provesti parsiranje koje će filtrirati suvišne *HTML* elemente, zadržavajući samo bitan tekstualni sadržaj, ključan za buduću obradu u strojno čitljiv opis. Elementi kao što su navigacija, *script*, *style*, zaglavlje, podnožje i mnogi drugi elementi koji su navedeni u nizu *elementsToDelete* često ne pružaju ključne informacije za generiranje *API* specifikacije, stoga se bez problema mogu izuzeti iz procesa. Funkcija *parseText* prima *HTTP* odgovor i izvodi sljedeće korake:

- Dohvaća sirovu *HTML* datoteku korištenjem *fetchText* metode.
- Pomoću *parseHTML* metode pretvara se sirova *HTML* datoteka u *DOM* objekt.

- Pomoću *deleteElementsByPartialName* metode uklanja se HTML elementi s imenima navedenim u *elementsToDelete* iz *DOM-a*.
- Na kraju, vraća sadržaj tijela dokumenta (tekstualni sadržaj) iz *DOM-a*.

```
    async parseText(response) {
            let rawData = await this.fetchText(response);
            const doc = this.parseHTML(rawData);
            const elementsToDelete = [
                "script", "nav", "sidenav", "topnav", "pagetop", "style",
                "header", "changelog", "navigation", "sidebar", "footer",
"toolbar",
                "aside", "banner", "donations"
            ];
          this.deleteElementsByPartialName(doc, elementsToDelete);

            return doc.body.textContent;
    }

    deleteElementsByPartialName(doc, names) {
            names.forEach(name => {
                const htmlElements = doc.querySelectorAll(`${name}`);
                const                    classElements                    =
doc.querySelectorAll(`[class*="${name}"]`);
                const                      idElements                     =
doc.querySelectorAll(`[id*="${name}"]`);
                const     elements     =     new     Set([...htmlElements,
...classElements, ...idElements]);

                elements.forEach(element => {
                    element.remove();
                });
          });
    }

    parseHTML(htmlString) {
            const dom = new JSDOM(htmlString);
            return dom.window.document;
    }

    async fetchText(response) {
```

```
            return     await     this.checkIfResponseIsOk(response)     ?
response.data : null;
      }

      async checkIfResponseIsOk(response){
              return await this.returnResponseCode(response) == 200;
      }

      async returnResponseCode(response){
             return response.status;
      }
```

5.3.5.Implementacija GPT modela

Nakon što se izbace nepotrebni *HTML* elementi, slijedi faza obrade tekstualnog dijela stranice dokumentacije. U toj fazi najveću ulogu ima *GPT* model, koji prima tekst i generira *OpenAPI* specifikaciju. Funkcija *getChatCompletionResponse* ima zadaću uspostaviti komunikaciju s *OpenAI* servisom kako bi dobila odgovor od *GPT-3* modela na temelju zadane chat poruke. Funkcija konkretno prima chat poruku kao argument (*chatPrompt)*. Pomoću *openai.chat.completions.create* metode, šalje zahtjev *OpenAI* servisu za generiranje odgovora na temelju zadane poruke. Prima odgovor od *OpenAI* servisa koji sadrži niz opcija za završavanje (*completion*) razgovora. Uzima se prvi element iz niza, koji predstavlja generirani odgovor.

```
      static async getChatCompletionResponse(chatPrompt) {
              try {
                  const completion = await openai.chat.completions.create({
                      model: 'gpt-3.5-turbo',
                      messages: [{ role: 'system', content: chatPrompt }],
                  });
                  const apiResponse = completion.choices[0].message.content;
                  return apiResponse;
              } catch (error) {
                  console.error('Greška prilikom slanja zahtjeva:', error);
                  return false;
              }
      }
```

5.3.5.1. Postavljanje autorizacijskog ključa za GPT model

Za optimalno funkcioniranje *GPT* modela i kako bismo osigurali sigurnost od potencijalne zloupotrebe, neophodno je postaviti autorizacijski ključ (koji smo prethodno dobili

od servisa) kao varijablu okruženja. U kontekstu naše web aplikacije, koja je razvijena na *Windows* operacijskom sustavu, postupak će biti sljedeći:

1. Kliknemo desnom tipkom miša na *Computer* (ili *This PC*) na radnoj površini i odaberemo *Properties*.
2. Kliknemo na *Advanced system settings* na lijevoj strani.
3. Kliknemo na *Environment Variables* donji dio prozora.
4. Pod sekcijom *User variables* ili *System variables*, kliknemo na *New* kako bismo dodali novu varijablu okruženja. Unesemo naziv varijable, koji će u našem slučaju biti *OPENAI_API_KEY* i njezinu vrijednost, koja je za svakog korisnika drugačija.

5.3.6. Generiranje interaktivnog sučelja dokumentacije

Kod ispod definira funkciju *getApiDocs* koja služi za generiranje i slanje *Swagger* dokumentacije *API-ja* kao odgovor na *HTTP* zahtjev. U prvom dijelu koda, putem *ds.readFileSync* čitaju se dijelovi *HTML* datoteka (*header*, *footer*, *downloadButton*) koji će biti uključeni u konačni odgovor. Ti dijelovi često predstavljaju vizualne elemente stranice, poput zaglavlja i podnožja. Zatim, funkcija pokušava čitati informacije o *API-ju* iz određene datoteke, koristeći *handler.readApiInfoFromFile.* Te informacije se koriste za generiranje *Swagger* dokumentacije. Nakon toga, stvaraju se opcije za *Swagger UI* (*options*) koje definiraju prilagodbe sučelja *Swagger* dokumentacije. Na primjer, skrivanje gornje trake (*customCss*) i postavljanje prilagođenog naslova stranice (*customSiteTitle*). Pomoću funkcije *swaggerUi.generateHTML* generira se *HTML* sadržaj za prikaz *Swagger* dokumentacije na web stranici. Konačni odgovor koji će biti poslan klijentu sastavljen je tako da se kombiniraju prethodno učitani dijelovi *HTML-a* (*header*, *Swagger UI*, *downloadButton*, *footer*) te se šalje kao odgovor na zahtjev.

```
exports.getApiDocs = async function (zahtjev, odgovor) {

    var header = ds.readFileSync(path.join(putanja, 
'html/partials/header.html')) + "<div class='container'>";

    var downloadButton = "</div>" + ds.readFileSync(path.join(putanja, 
'html/components/download-button.html'));

    var footer = ds.readFileSync(path.join(putanja, 
'html/partials/footer.html'));


    try {
        const apiInfo = await handler.readApiInfoFromFile(filePath);
        const swaggerDocument = apiInfo;
        const options = {
          customCss: '.swagger-ui .topbar { display: none }',
          customSiteTitle: 'API documentation'
```

```
            }
            const swaggerUiHtml = swaggerUi.generateHTML(swaggerDocument,
options);
            odgovor.send(header + swaggerUiHtml + downloadButton + footer);

          } catch (error) {
            console.error('Greška prilikom generiranja Swagger
dokumentacije:', error);
            odgovor.redirect(`/errorPage?error=500`);
          }
      }
```

5.3.6.1. Preuzimanje dokumentacije

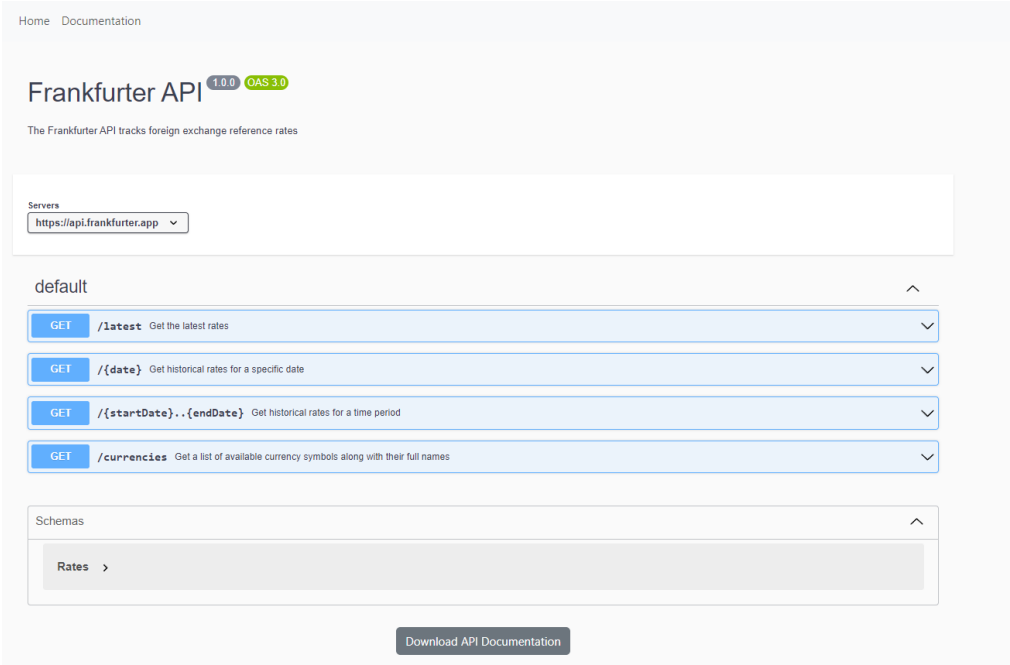
Ako smo zadovoljni generiranom *API* dokumentacijom, imamo mogućnost da je preuzmemo sa iste stranice na kojoj je prikazana. To možemo postići tako što kliknemo na gumb *Download API Documentation* koji je smješten na donjem dijelu stranice.

```
      exports.downloadApiDocs = async (zahtjev, odgovor) {
        const openApiSpec = ds.readFileSync(filePath, 'utf8');
        odgovor.setHeader('Content-disposition', 'attachment; filename=api-
info.json');
        odgovor.setHeader('Content-type', 'application/json');
        odgovor.send(openApiSpec);
      }
```

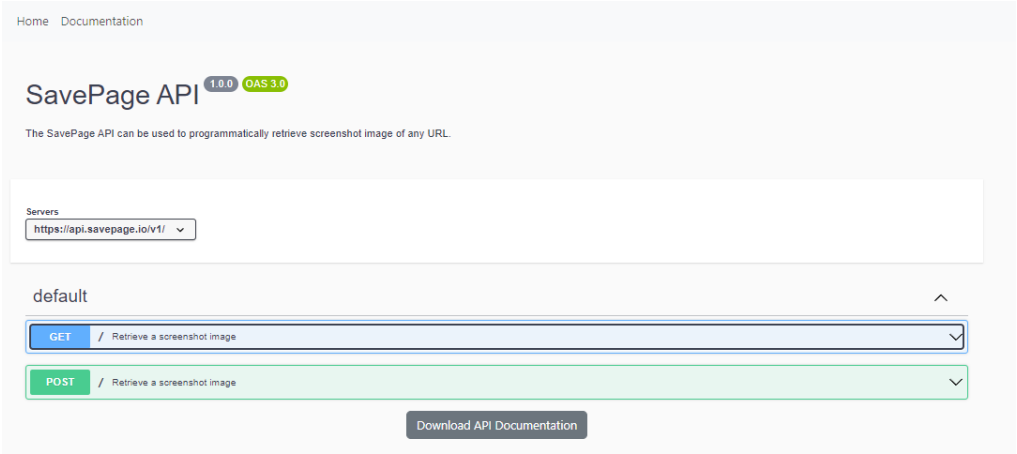
5.4. Testiranje generiranja specifikacije nad stvarnim dokumentacijama

Sada putem web sučelja imamo mogućnost izvršiti testiranje procesa generiranja specifikacija. U ovom konkretnom slučaju, fokusiramo se na testiranje na primjeru *Frankfurter API-ja* [16], koji smo ranije predstavili kao primjer *API* specifikacije bez formalnog opisa u dokumentaciji, ali će sam proces generiranja specifikacije biti prikazan i na drugim API-jima. Ovo testiranje je od iznimne važnosti kako bismo potvrdili točnost generiranja specifikacija. Uporaba stvarnih dokumentacija omogućava nam da osiguramo da su generirane specifikacije precizne i da odgovaraju stvarnim podacima. Posebna važnost leži u validaciji funkcionalnosti, jer provjeravamo jesu li svi ključni dijelovi dokumentacije ispravno preneseni u standardiziranu specifikaciju, kao što su putanje resursa, parametri, primjeri odgovora i drugi detalji. Treba napomenuti da je mogućnost testiranja ograničena na *API-je* manjeg opsega, što je jedno od ograničenja naše aplikacije. S obzirom na činjenicu da GPT model nije dostupan besplatno, izrada ove specifikacije nas je stajala manje od jednog centa. U usporedbi s potencijalnim

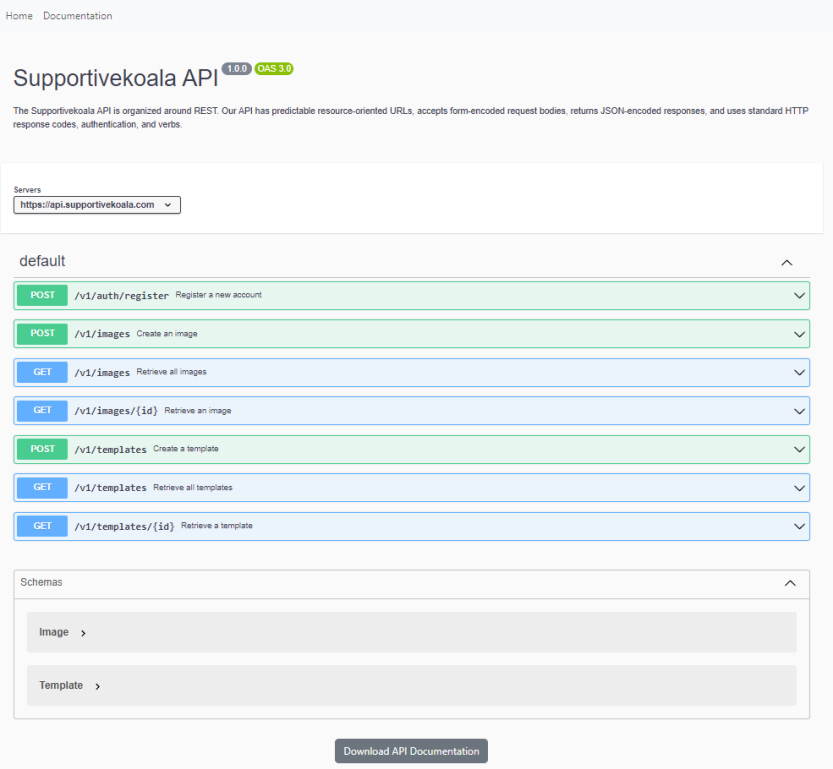
vremenom i financijskim troškovima koje bi zahtijevalo ručno generiranje specifikacije unutar neke kompanije, ova automatizirana metoda je znatno ekonomičnija. Uobičajeno bi takav proces trajao nekoliko sati i rezultirao s nekoliko eura više potrošenih resursa.



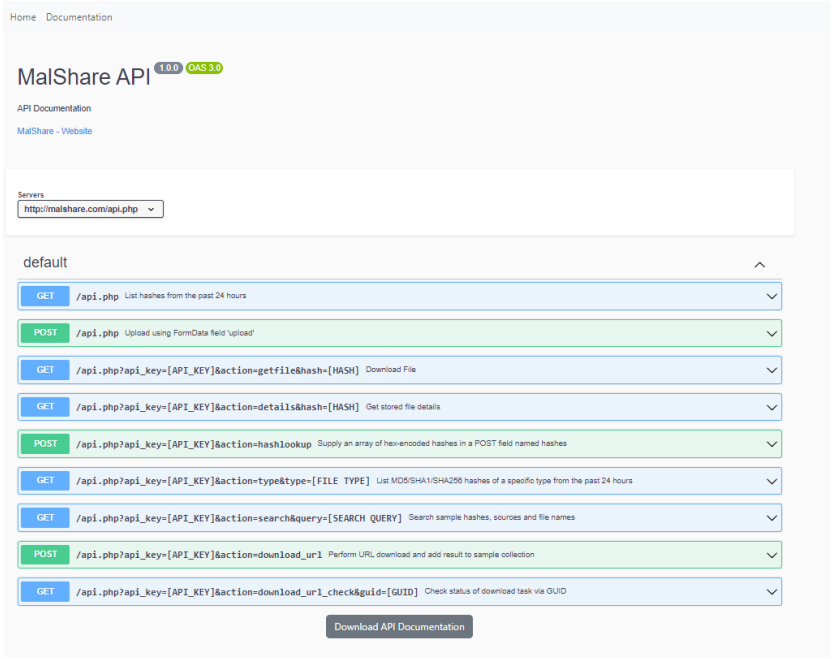
Slika 21: Prikaz interaktivno generirane specifikacije Frankfurter API [autorski rad]



Slika 22: Prikaz interaktivno generirane specifikacije SavePage API [autorski rad]



Slika 23: Prikaz interaktivno generirane specifikacije SupportiveKoala API [autorski rad]



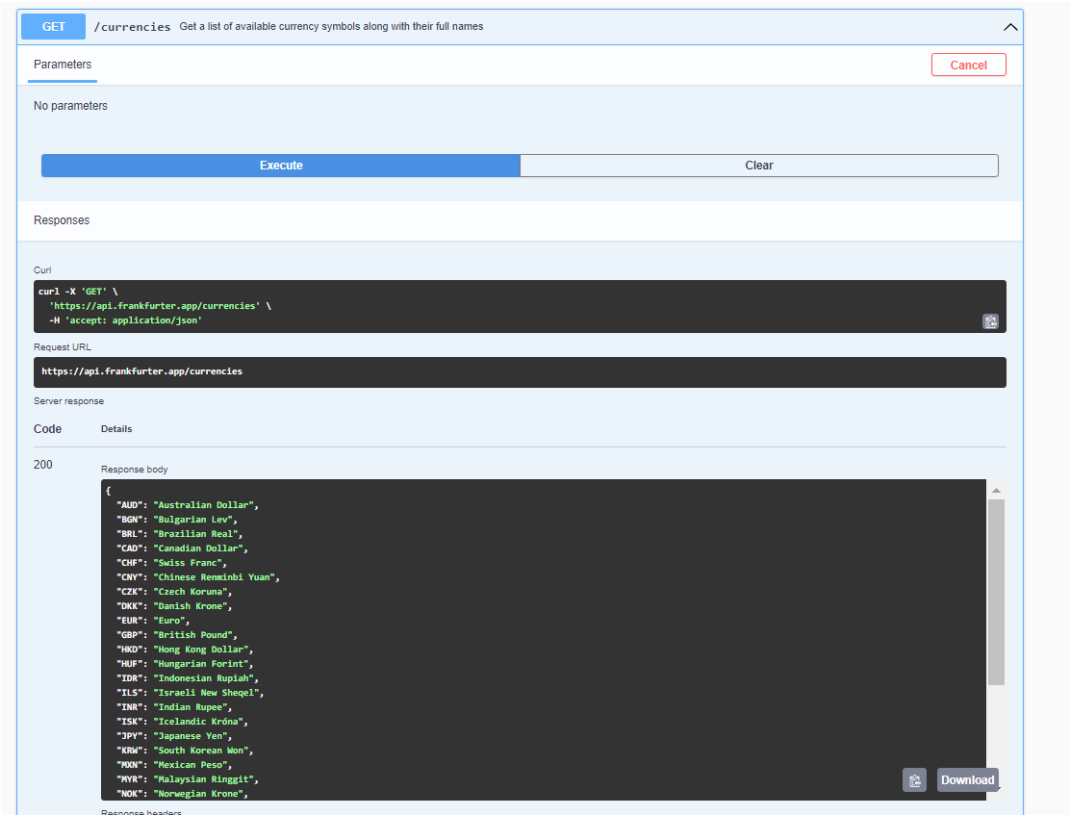
Slika 24: Prikaz interaktivno generirane specifikacije MalShare API [autorski rad]

Uz pomoć aplikacije, generiranje specifikacije testirano je nad sljedećim API-jima: *Chuck Norris Jokes Api* [19], *Digimon API* [20], *Disney API* [21], *SupportiveKoala* [22], *Free Currency API* [23], *Free-To-Play Games Database API* [24], *Free Games, Keys & Giveaways*

Tracker API [25], *Game of Thornes Quotes API* [26], *MMO Games & MMORPG API* [27], *MalShare* [28], *RPS-101* [29], *SavePage API* [30], *shrtlnk API* [31].

5.5. Testiranje ispravnosti API-ja preko SwaggerUI

Testiranje ispravnosti igra ključnu ulogu u razvoju i održavanju web aplikacija, pružajući niz značajnih prednosti. *Swagger UI*, kao alat za testiranje, omogućuje interaktivno sučelje koje pojednostavljuje proces provjere generirane *API* specifikacije. Ovaj alat omogućuje brz i jednostavan način za generiranje i slanje zahtjeva, čime se eliminira potreba za ručnim stvaranjem zahtjeva. *Swagger UI* čak pruža i mogućnost generiranja *curl* naredbi koje se mogu testirati izravno u terminalu, nakon instalacije *curl* alata. Sljedeća slika prikazuje odgovor na zahtjev koji je generiran za resurs */currencies*. On konkretno vraća popis svih dostupnih simbola valuta, zajedno s njihovim kraćim i punim nazivom.



Slika 25: Prikaz odgovora na zahtjev za odabrani resurs Frankfurter API-ja [autorski rad]

5.6. Prednosti i ograničenja aplikacije

Realizirana aplikacija koja se bavi automatskim generiranjem opisa web servisa u obliku *OpenAPI* specifikacije donosi određene prednosti i ograničenja:

Prednosti:

- **Standardizacija**: *OpenAPI* specifikacija pruža standardiziran format za dokumentaciju *API-ja*, što olakšava razumijevanje i komunikaciju između različitih timova i razvojnih procesa.
- **Efikasnost**: Automatsko generiranje *OpenAPI* specifikacije olakšava i ubrzava proces dokumentiranja web servisa. Manualno pisanje specifikacija može biti vremenski zahtjevno, dok automatska generacija osigurava konzistentnost i brži razvoj.
- **Automatsko testiranje preko interaktivnog sučelja**: Mogućnost da korisnici odrade brzu i učinkovitu provjeru ispravnosti *API* specifikacije.

Ograničenja:

- **Ograničenja web struktura**: Aplikacija može naići na probleme kod dohvaćanja informacija s web stranica koje imaju nestandardnu ili dinamičku strukturu.
- **Ovisnost o vanjskim servisima**: Aplikacija može biti ovisna o funkcionalnosti vanjskih servisa, poput *GPT* modela za generiranje teksta. Ako ti servisi nisu dostupni ili se mijenjaju, to može utjecati na funkcionalnost generacije specifikacija.
- **Ograničen broj tokena u komunikaciji**: U sadašnjem stanju, *GPT-3* pruža ograničenje od 16 tisuća tokena po jednom zahtjevu, što uključuje i pitanje i odgovor. To ograničava mogućnost testiranja aplikacije na većim *API-ima*. Iako je teoretski moguće razbiti pitanje na više uzastopnih poruka, takav pristup može dovesti do gubitka konteksta u drugim porukama, stvarajući izazov u očuvanju sveobuhvatnog razumijevanja.

6. Zaključak

Koncept servisno usmjerenih arhitektura (SOA) potiče razvoj i upotrebu neovisnih servisa koji su fleksibilni i pružaju ponovnu iskoristivost. Među tim servisima, Web servisi igraju ključnu ulogu u digitalnom okruženju, olakšavajući svakodnevne procese preko mreže. Razvoj i podržavanje web servisa ima izuzetan značaj, a dodatnu vrijednost donosi standardizirani opis ili specifikacija koja pomaže u maksimalnom iskorištavanju njihovih operativnih karakteristika.

Naša implementirana web aplikacija za analizu RESTful web servisa omogućuje automatsko generiranje opisa web servisa, što može biti korisno u različitim fazama razvoja. RESTful servisi su jednostavni, fleksibilni i efikasni, a njihovi odgovori u JSON formatu su čitljivi i ne zahtijevaju dodatnu obradu.

Iako trenutno postoji ograničenje u analizi većih RESTful web servisa zbog ograničenja GPT-3 na 16 tisuća tokena, ova aplikacija prikazuje potencijal takvog pristupa i ulogu umjetne inteligencije u automatizaciji procesa razvoja web servisa. Ovakav oblik automatizacije smanjuje potrebu za intervencijom ljudskih resursa, čime se smanjuju troškovi. Iako korištenje GPT modela ima svoj trošak, brza i precizna izrada specifikacije za svega nekoliko centi značajno nadmašuje vrijeme i trošak koji bi bio potreban za ručnu izradu.

Sve više i više se prepoznaje potencijal obrade prirodnog jezika poput GPT-a, a broj aplikacija koje automatiziraju različite procese sve više raste. Primjerice, danas možemo dobiti visokokvalitetne slike u nekoliko sekundi, što bi inače zahtijevalo satima ili čak danima truda umjetnika. Ovaj primjer ilustrira kako automatizacija pomoću GPT modela donosi učinkovitost, brzinu i ekonomičnost u različitim industrijama.

Popis literature

[1.] T. Erl, Paulo Merson, and R. Stoffers, Service-oriented architecture : analysis and design for services and microservices. Boston: Prentice Hall, 2017.

[2.] M. Papazoglou, Web services : principles and technology. Harlow, England ; New York: Pearson/Prentice Hall, 2008.

[3.] A. Walker, „What is SOA? Service-Oriented Architecture Principles“, 2023. [Blog post]. Dostupno: https://www.guru99.com/soa-principles.html [pristupano 08.08.2023].

[4.] „RESTful Web Services“ (22.12.2022.). GeeksforGeeks [Na internetu]. Dostupno: https://www.geeksforgeeks.org/restful-web-services/ [pristupano 08.08.2023].

[5.] „Types of Web Services“ (bez dat.). javatpoint [Na internetu]. Dostupno: https://www.javatpoint.com/restful-web-services-types-of-web-services [pristupano 08.08.2023].

[6.] L. Gupta, „Statelessness in REST APIs“, 2022. [Blog post]. Dostupno: https://restfulapi.net/statelessness/ [pristupano 08.08.2023].

[7.] A. Walker, „SOAP Web Services Tutorial: What is SOAP Protocol? EXAMPLE“, 2023. [Blog post]. Dostupno: https://www.guru99.com/soap-simple-object-access-protocol.html [pristupano 08.08.2023].

[8.] „What is Natural Language Processing? Definition and Examples“ (16.06.2023). Coursera [Na internetu]. Dostupno: https://www.coursera.org/articles/natural-language-processing [pristupano 08.08.2023].

[9.] „What Is Natural Language Processing (NLP)?“ (bez dat.). Amazon Web Services, Inc. [Na internetu]. Dostupno: https://aws.amazon.com/what-is/nlp/ [pristupano 08.08.2023].

[10.] R. Wolff, „Natural Language Processing (NLP): 7 Key Techniques“, 2021. [Blog post] Dostupno: https://monkeylearn.com/blog/natural-language-processing-techniques/ [pristupano 08.08.2023].

[11.] P. Orza, „11 Real-Life Examples of NLP in Action“, 2022. [Blog post]. Dostupno: https://levity.ai/blog/11-nlp-real-life-examples [pristupano 08.08.2023].

[12.] „How eCommerce uses Natural Language Processing (NLP) in 2022“ (bez dat.). BroutonLab [Blog post]. Dostupno: https://broutonlab.com/blog/nlp-and-ai-for-ecommerce [pristupano 08.08.2023].

[13.] A. Hutanu, „How ChatGPT works and AI, ML & NLP Fundamentals“, 2023. [Blog post]. Dostupno: https://www.pentalog.com/blog/tech-trends/chatgpt-fundamentals/ [pristupano 08.08.2023].

[14.] „WSDL - Quick Guide“ (bez dat.). Tutorials point [Na internetu]. Dostupno: https://www.tutorialspoint.com/wsdl/wsdl_quick_guide.htm [pristupano 08.08.2023].

[15.] „OpenAPI Specification“ (bez dat.). Swagger [Na internetu], Dostupno: https://swagger.io/specification/ [pristupano 08.08.2023].

[16.] „Documentation“ (bez dat.). Frankfurter [Na internetu], Dostupno: https://www.frankfurter.app/docs/ [pristupano 16.08.2023].

[17.] P. Merrill, „Chomp Food API“. [Na internetu]. (bez dat.). Dostupno: https://chompthis.com/api/ [pristupano 16.08.2023].

[18.] „About npm“ (bez dat.). npm [Na internetu]. Dostupno: https://www.npmjs.com/about [pristupano 16.08.2023].

[19.] „Chuck Norris Jokes Api“ (bez dat.). api.chucknorris.io [Na internetu]. Dostupno: https://api.chucknorris.io/ [pristupano 17.08.2023].

[20.] „Digimon API“ (bez dat.). digimon-api.vercel.app [Na internetu]. Dostupno: https://digimon-api.vercel.app/ [pristupano 17.08.2023].

[21.] „Documentation“ ,2021. disneyapi.dev [Na internetu]. Dostupno: https://disneyapi.dev/docs/ [pristupano 17.08.2023].

[22.] „Documentation“ (bez dat.). developers.supportivekoala.com [Na internetu]. Dostupno: https://developers.supportivekoala.com/ [pristupano 17.08.2023].

[23.] „Free Currency API“ (bez dat.). Amdoren [Na internetu]. Dostupno: https://www.amdoren.com/currency-api/ [pristupano 17.08.2023].

[24.] „Free-To-Play Games Database API“ (bez dat.). freetogame [Na internetu]. Dostupno: https://www.freetogame.com/api-doc [pristupano 17.08.2023].

[25.] „Free Games, Keys & Giveaways Tracker API“ (bez dat.). gamerpower [Na internetu], Dostupno: https://www.gamerpower.com/api-read [pristupano 17.08.2023].

[26.] „Game of Thrones Quotes API“ (bez dat.). Game of Thrones Quotes API [Na internetu], Dostupno: https://gameofthronesquotes.xyz/ [pristupano 17.08.2023].

[27.] „MMO Games API - By MMOBomb“ (bez dat.). MMOBomb [Na internetu]. Dostupno: https://www.mmobomb.com/api [pristupano 17.08.2023].

[28.] „API Documentation“ (bez dat.). MalShare [Na internetu]. Dostupno: https://malshare.com/doc.php [pristupano 17.08.2023].

[29.] „RPS-101“ (bez dat.). rps101.pythonanywhere.com [Na internetu]. Dostupno: https://rps101.pythonanywhere.com/api [pristupano 17.08.2023].

[30.] „Documentation“ (bez dat.). SavePage API [Na internetu]. Dostupno: https://docs.savepage.io/ [pristupano 17.08.2023].

[31.] „Shrtlnk API Documentation“ (bez dat.). shrtlnk [Na internetu]. Dostupno: https://www.shrtlnk.dev/developer/documentation [pristupano 17.08.2023].

[32.] D. Andročec, „Servisno orijentirana arhitektura za Internet stvari,“ nastavni materijali na kolegiju Servisi Internet stvari [Moodle], Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin, 2023.

[33.] Node.js [Slika] (bez dat.). Dostupno: https://en.wikipedia.org/wiki/Node.js [pristupano 18.08.2023.]

[34.] „Pricing“ (bez dat.). OpenAI [Na internetu]. Dostupno: https://openai.com/pricing. [pristupano 18.08.2023.]

[35.] M. Nehra, „Key Benefits of Service Oriented Architecture“, 2023. [Blog post]. Dostupno: https://www.decipherzone.com/blog-detail/service-oriented-architecture [pristupano 21.08.2023.]

[36.] „What are the challenges in SOA?“ (bez dat.). Ques10 [Na internetu]. Dostupno: https://www.ques10.com/p/338/what-are-the-challenges-in-soa-3/ [pristupano 21.08.2023.]

[37.] „Web Services Tutorial: Components, Architecture, Type & Examples“ (23.06.2023). Software Testing Help [Na internetu]. Dostupno: https://www.softwaretestinghelp.com/web-services-tutorial/ [pristupano: 21.08.2023.]

[38.] „Web Services - Architecture“ (bez dat.). Tutorialspoint [Na internetu]. Dostupno: https://www.tutorialspoint.com/webservices/web_services_architecture.htm [pristupano 21.08.2023].

[39.] „REST API Architectural Constraints“ (09.12.2018.). GeeksforGeeks [Na internetu]. Dostupno: https://www.geeksforgeeks.org/rest-api-architectural-constraints/ [pristupano 21.08.2023].

[40.] „About Node.js“ (bez dat.). Node.js [Na internetu]. Dostupno: https://nodejs.org/en/about [pristupano 21.08.2023].

[41.] E. L. Spencer, „Natural Language Processing (NLP) Techniques & Examples“, 2021. [Blog post]. Dostupno: https://www.revuze.it/blog/natural-language-processing-techniques/ [pristupano 21.08.2023].

[42.] „The structure of a SOAP message,“ (19. 06. 2023.). IBM [Na internetu]. Dostupno: https://www.ibm.com/docs/en/integration-bus/10.1?topic=soap-structure-message [pristupano 21.08.2023].

[43.] „TOP 7 REST API Security Threats,“ (09.01.2019). RestCase [Blog post]. Dostupno: https://blog.restcase.com/top-7-rest-api-security-threats/ [pristupano 21.08.2023].

[44.] „XML WSDL,“ (bez dat.). w3schools [Na internetu]. Dostupno: https://www.w3schools.com/xml/xml_wsdl.asp [pristupano 21.08.2023].

[45.] R. Awati, „UDDI (Universal Description, Discovery and Integration)“, 2023. [Blog post]. Dostupno: https://www.techtarget.com/whatis/definition/UDDI-Universal-Description-Discovery-and-Integration [pristupano 22.08.2023].

[46.] „UDDI - Quick Guide,“ (bez dat.). Tutorialspoint [Na internetu]. Dostupno: https://www.tutorialspoint.com/uddi/uddi_quick_guide.htm [pristupano 22.08.2023].

[47.] „UDDI Examples,“ (bez dat.). Cs.au.dk [Na internetu]. Dostupno: https://cs.au.dk/~amoeller/WWW/webservices/uddiexamples.html [pristupano 22.08.2023].

[48.] „WADL documents,“ (09.03.2021.). IBM [Na internetu]. Dostupno: https://www.ibm.com/docs/en/rtw/9.2.0?topic=testing-wadl-documents [pristupano 22.08.2023].

[49.] „WADL – Web Application Description Language,“ (bez dat.). wallarm [Na internetu]. Dostupno: https://www.wallarm.com/what/what-is-wadl [pristupano 22.08.2023].

[50.] „Example: The Storage-Service WADL,“ (bez dat.). docs.oracle.com [Na internetu]. Dostupno: https://docs.oracle.com/cd/E19776-01/820-4867/ghmfe/ [pristupano 22.08.2023].

[51.] „Semantic web service,“ (19.08.2023). Wikipedia, the Free Enyclopedia [Na internetu]. Dostupno: https://en.wikipedia.org/wiki/Semantic_web_service [pristupano 22.08.2023].

[52.] „SAWSDL“, (bez dat). W3 Semantic Web [Na internetu]. Dostupno: https://www.w3.org/2001/sw/wiki/SAWSDL [pristupano 22.08.2023].

[53.] K.Gomadan, A.Ranabahu, A.Sheth, „SA-REST: Semantic Annotation of Web Resources“, 2010. [Na internetu]. Dostupno: https://www.w3.org/Submission/SA-REST/ [pristupano 22.08.2023].

[54.] „OWL,“ (bez dat.). W3 Semantic Web [Na internetu]. Dostupno: https://www.w3.org/OWL/ [pristupano 22.08.2023].

[55.] „RDF,“ (bez dat.). W3 Semantic Web [Na internetu]. Dostupno: https://www.w3.org/2001/sw/wiki/RDF [pristupano 22.08.2023].

[56.] J. Chen, „Neural Network Definition“, 2022. [Na internetu]. Dostupno: https://www.investopedia.com/terms/n/neuralnetwork.asp [pristupano 22.08.2023].

[57.] R. Vidiyala, „6 Types of Neural Networks Every Data Scientist Must Know“, 2020. [Na internetu]. Dostupno: https://towardsdatascience.com/6-types-of-neural-networks-every-data-scientist-must-know-9c0d920e7fce [pristupano 23.08.2023].

[58.] „What is Deep Learning?,“ (bez dat.). Amazon Web Services, Inc. [Na internetu]. Dostupno: https://aws.amazon.com/what-is/deep-learning/ [pristupano 23.08.2023].

[59.] „Deep Learning vs. Neural Networks,“ (26.10.2022). PureStorage [Blog post]. Dostupno: https://blog.purestorage.com/purely-informational/deep-learning-vs-neural-networks/ [pristupano 23.08.2023].

[60.] A. Mehra, „A Deep Dive into GPT Models: Evolution & Performance Comparison“, 2023. [Blog post]. Dostupno: https://www.kdnuggets.com/2023/05/deep-dive-gpt-models.html [pristupano 23.08.2023].

[61.] M. Diaz, „How to use Bing Chat (and how it's different from ChatGPT)“ , 2023. [Na internetu]. Dostupno: https://www.zdnet.com/article/how-to-use-the-new-bing-and-how-its-different-from-chatgpt/ [pristupano 23.08.2023].

[62.] „Bard (bez dat.). Bard [Na internetu]. Dostupno: https://bard.google.com/ [pristupano 23.08.2023].

[63.] „What is Perceptron: A Beginners Guide for Perceptron“ [Slika] (bez dat.). Dostupno: https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron [pristupano 23.08.2023.]

[64.] „What is multilayer perceptron?“ [Slika] (17.12.2022). Dostupno: https://www.nomidl.com/natural-language-processing/what-is-multilayer-perceptron/ [pristupano 23.08.2023.]

[65.] „Convolutional Neural Network: An Overview“ [Slika] (27.01.2022). Dostupno: https://www.analyticsvidhya.com/blog/2022/01/convolutional-neural-network-an-overview/ [pristupano 23.08.2023.]

[66.] „Overview of GAN Structure“ [Slika] (bez dat.). Dostupno: https://developers.google.com/machine-learning/gan/gan_structure [pristupano 23.08.2023.]

Popis slika

Slika 1: Arhitektura web servisa 7
Slika 2: Struktura SOAP poruke 12
Slika 3: Perceptron model 28
Slika 4: Model Više slojnih perceptrona 28
Slika 5: Model Konvoluncijskih neuronskih mreža 29
Slika 6: Model Rekurzivne neuronske mreže 30
Slika 7: Model Mreža dugoročne kratkoročne memorije 30
Slika 8: Model Generativne suparničke mreže 31
Slika 9: Pregled ChatGPT na stolnom računalu 34
Slika 10: Pregled Bing Chat-a na stolnom računalu 37
Slika 11: Pregled Bard AI na stolnom računalu 37
Slika 12: Pregled Frankfurter API-ja 38
Slika 13: Pregled resursa Chomp Food & Recipe API-ja 39
Slika 14: Pregled resursa koji se bavi dohvaćanjem hrane prikazan 39
Slika 15: Prikaz polja za unos parametara odabranog resursa 40
Slika 16: Prikaz primjera odgovora na pozivanje zadanog resursa na stolnom računalu 41
Slika 17: Sučelje Visual Studio Code-a 42
Slika 18: Logo Node.js-a 43
Slika 19: Pregled Node.js početne stranice na stolnom računalu 43
Slika 20: Prikaz forme za unos putanje do dokumentacije 45
Slika 21: Prikaz interaktivno generirane specifikacije Frankfurter API 50
Slika 22: Prikaz interaktivno generirane specifikacije SavePage API 50
Slika 23: Prikaz interaktivno generirane specifikacije SupportiveKoala API 51
Slika 24: Prikaz interaktivno generirane specifikacije MalShare API 51
Slika 25: Prikaz odgovora na zahtjev za odabrani resurs Frankfurter API-ja 52

Prilozi

[1.] GitHub repozitorij web aplikacije