Interoperabilnost Web API-a

Matić, Mateo

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mateo Matić

INTEROPERABILNOST WEB API-a

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mateo Matić

Matični broj: 41374/12-IZV

Studij: Informacijski sustavi

INTEROPERABILNOST WEB API-a

ZAVRŠNI RAD

Mentor/Mentorica:

Doc. dr. sc. Andročec Darko

Varaždin, Rujan 2020.

Mateo Matić

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

U ovom radu proći ćemo kroz osnovne pojmove kao što su interoperabilnost i Web API od čega se sastoje i koji se koriste u svakodnevnici. Polazi se kroz strukturu Web API-a i njegove varijante kao što je REST API, njime se želi prikazati poboljšana verzija Web API-a koja ima svoje predefinirane standarde i strukturu. Pritom ću prikazati njegove prednosti i mane. U praktičnom dijelu gdje povezujemo dvoje Web API-a (SOAP i REST API), te ih potom prikazujemo u njihovom radu. Te nakraju zaključujemo po čemu je koja verzija Web API-a bolja i zašto bi je trebali koristiti u bilo kojem projektu ili većoj organizaciji.

Ključne riječi: Interoperabilnost, SOAP API, REST API, Mikroservisi, Web servisi, Web aplikacija, XML, JSON

Sadržaj

1. Uvod

Tema ovog završnog rada je interoperabilnost Web API-a i njezini problemi na koje nailazimo tokom implementacije same. Sama interoperabilnost je mogućnost neke aplikacije ili sustava da podržava razmjenu podataka ili informacija.

Zatim govorimo koji su propusti kod Web API-a i koji se problemi javljaju s njime. Motivacija pisanja završnog rada je prikaz tih problema ali tako i rješenja koja se mogu riješiti koristeći RESTful API koji ima standardizirane protokole kod implementacije.

Neki od problema interoperabilnosti API-a su nedostatak podatkovnih standarda kojima se šalju podaci kao i sama struktura API-a koja se mora krojiti po servisu koji će je koristiti. Takvom izgradnjom dolazi do problema kreiranje nekolicine API-a različitih namjera, to dovodi do još jednog većeg problema koji se odnosi na nadogradnju sustava i promjene spremišta podataka koji iza sebe ostavlja veliku količinu posla kod nadogradnje samog Web API-a.

2. Metode i tehnike rada

Kao praktični dio napravio sam prikaz interoperabilnosti dvoje Web API-a:

- **SOAP API**
- **REST API**

Njihova komunikacija je ostvarena tako da REST API se poziva svojim zahtjevom koji potom u pozadini preuzima podatke od SOAP API-a, potom će biti prikazani neki od problema Web API-a.

Neki od problema koje sam prikazao su:

1. Korištenje GET zahtjeva umjesto HTTP POST
2. Različiti formati slanja i primanja podataka
3. Autentifikacija zahtjeva
4. Problem ne pokrivanja većine statusnih kodova koje API vraća

Alate koje sam koristio pri izradi tehničkog dijela završnog rada su:

- PHP
- HTML
- CSS
- JavaScript/Jquery
- Postman

3. Interoperabilnost

Interoperabilnost je karakteristika nekog sustava koji sadrži sučelja za komunikaciju sa ostalim sustavima. Postoje mnoge definicije interoperabilnosti sustava ali nabrojati ću neke od najvažnijih svjetskih organizacija.

Definicije interoperabilnosti:

- **Webster online rječnik** – navodi interoperabilnost kao „sposobnost razmjene i korištenje informacija unutar najčešće lokalne mreže [7].
- **Ministarstvo obrane (DoD)** – definira interoperabilnost kao „ispunjeni uvjet između komunikacijsko-elektroničkih sustava ili dio komunikacijsko-elektroničke opreme kada se informacije ili servisi mogu izravno i uspješno razmjenjivati između njih i/ili krajnjih korisnika. Oni navode da se stupanj interoperabilnosti treba definirati kod specifičnih slučajeva [7].
- **Institut inženjera i elektrotehnike i elektronike (IEEE)** – govori o interoperabilnosti kao „sposobnost dva ili više sustava da razmjenjuju informacije i koriste informacije koje su razmijenjene“ [7].
- **Međunarodna organizacija za standardizaciju / Međunarodna elektrotehnička komisija (ISO/IEC)** – definira interoperabilnost kao „sposobnost komunikacije, izvršavanje programa ili prijenosa podataka između različitih funkcionalnih jedinica na način koji zahtjeva da korisnik ima malo ili nimalo znanja o jedinstvenim karakteristikama tih jedinica [7].

Gore navedene definicije nisu formalne ali naglašavaju dvije točke koje se ponavljaju u svim definicijama, to se odnosi na:

- Razmjenu informacija
- Korisnost informacija

3.1. Vrste interoperabilnosti

Postoje tri vrste interoperabilnost i svaki je jedinstven na svoj način, a to su:

- Sintaksična interoperabilnost
- Semantička interoperabilnost
- Međudomenska interoperabilnost

Kada govorimo o sintaksičnoj interoperabilnost to se odnosi na najjednostavniji oblik interoperabilnosti koji reprezentira dva ili više sustava koji imaju mogućnost komuniciranja između sebe, a tako i korištenje uobičajenih formata podataka i komunikacijskih protokola [8]. Dobar primjer koji bi mogao prikazati ovu vrstu je korištenje XML-a i SQL-a kao uobičajen format i protokol sintaktična značenje komunikacije između sustava. Niže razine formata podataka također doprinose sintaksičnoj interoperabilnost tako što osiguravaju pohranu podataka u istom ASCII ili Unicode formatima kod svih komunikacijskih sustava.

Semantička interoperabilnost nam predstavlja automatsko razumijevanje informacijske razmjene i točnosti kako bi se dobio uspješan rezultat koji je definiran od strane krajnjeg korisnika ili oba sustava [8]. Da bi se to moglo ostvariti oba sustava moraju dogovoriti točne specifikacije oko modela razmjene informacija, što znači da informacije koje se šalju jednake su onim informacijama koje se zaprimaju.

Međudomenska interoperabilnost nam uključuje više sustava bilo to društvenih, organizacijskih, političkih ili pravnih elemenata koji rade zajedno zbog zajedničkih interesa i/ili razmjene informacija.

3.2. Problemi interoperabilnosti

Problemi koji se mogu javiti prilikom kreiranja interoperabilnog sustava možemo razdijeliti na četiri potencijalna problema:

1. **Sistemska pogreška** – kada govorimo o potencijalnoj sistemskom problemu to se odnosi na nepodudarnost između serverskih komponenti i operacijskog sustava, a tako i na tehničku razmjenu podatka kroz mrežu, samih računala i web servise [9].

2. **Sintaktična pogreška** – ona direktno utječe na sposobnost interoperabilnosti sustava jer se reprezentira kao razlika u kodiranju, dekodiranju i prikazu podatka [9].

3. **Strukturalna pogreška** – odnosi se na informacijske arhitektonske varijante u podatkovnim modelima, strukturama i shemama. Zbog svojih različitosti one nadodaju još jedan sloj interoperabilnih prepreka [9].

4. **Semantička pogreška** – odnosi se na probleme koji se mogu pojaviti ako se unaprijed ne definiraju podaci koji će se razmjenjivati, jer se njihove sličnosti, razlike i međusobni odnosi moraju razumjeti kako bi se interoperabilnost sustava mogla ostvariti [9].

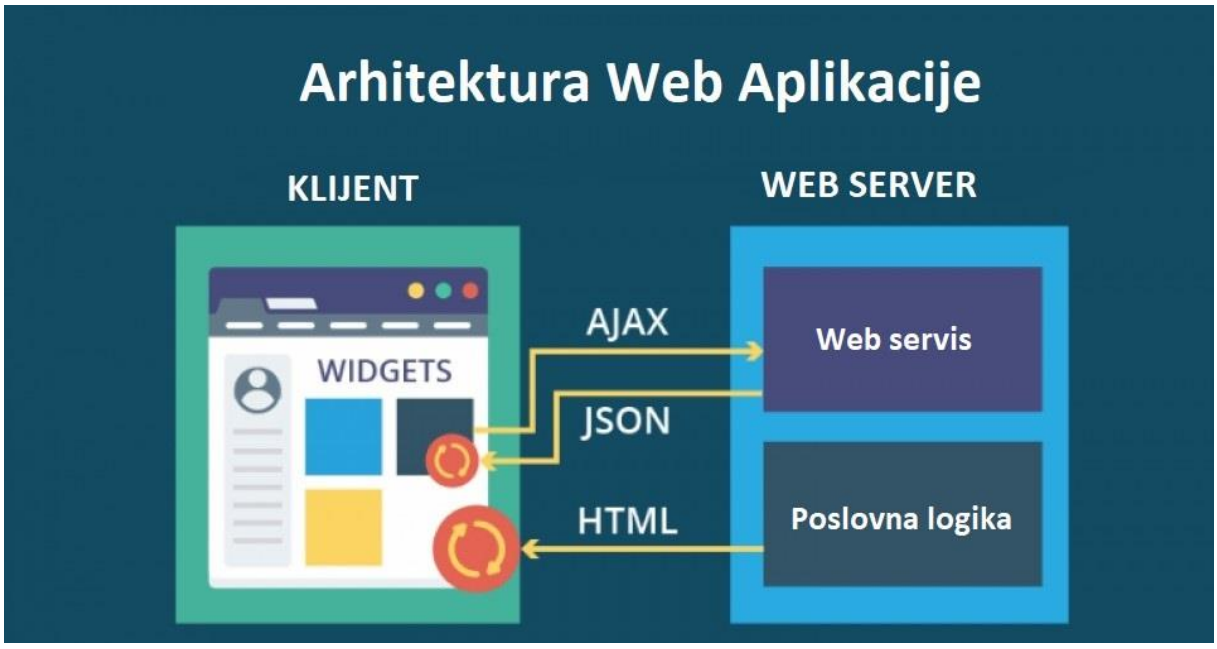
4. Web aplikacije

Web aplikacija je aplikacijski program ili skupina programa koji omogućuju procesuiranje informacija, takav zahtjev za procesuiranje se započinje preko internetskog preglednika ali se samo procesuiranje odvija na Web serveru, aplikacijskom serveru i/ili bazi podataka [1].

Kada govorimo o komponentama web aplikacija one se sastoje od:

- UI/UX web aplikacijskih komponenti
- Strukturne komponente
- Klijentska komponenta
- Serverska komponenta

Svaka datoteka pohranjena na serverskoj komponenti predstavlja niz sadržaja i instrukcija za formatiranje prikaza na klijentskoj strani u HTML jeziku. Većina web stranica na klijentskoj komponenti sadrže skripte koje se interpretiraju u web pregledniku i nadodaju dinamičnosti web stranice. Krajnjem korisniku omogućuje se interakcija sa samom web aplikacijom, a za sami pristup web aplikacijama potrebni su nam: „kontinuirani pristup internetu“ i „web preglednik“(Google Chrome, Mozilla Firefox, Safari, Opera).



Slika 1. Arhitektura Web Aplikacije [20]

4.1. Povijest razvoja web aplikacija

Povijest web aplikacije kreće u ranim početcima interneta, svaka web stranica je bila dostavljena do klijentskog računala kao statični dokument. Sekvenca takvih dokumenata je mogla pružiti interaktivno iskustvo korisniku kroz ugrađene web forme u dokumentima. Kod svake važnije promjene podaci su se morali ponovo preuzeti s web servera.

Sve je krenulo 1995. godine kada je Netscape izumio JavaScript skripti programski jezik koji se izvršava na klijentskoj strani i pruža potpunu dinamičnost web stranica. Programerima je to omogućilo dodavanje dinamičnih elemenata korisničkom sučelju. Umjesto slanja podataka na server kako bi se izgenerirala cijela web stranica ugrađene skripte mogu odraditi različite zadatke kao što je validacija korisničkog unosa i prikazivanje sakrivenih dijelova stranice [3].

1996. godine Macromedia je predstavila Flash koji je pomoću vektorskih animacija omogućio dodavanje plugin-ova kako bi se dobila fluidnost animacija kod navigiranja web stranicom. Također je omogućilo skriptnim jezicima da klijentska strana može komunicirati bez potrebe da se ponovo dohvaćaju podaci sa servera [3].

1999. godine koncept web aplikacija predstavljen je u Java programskom jeziku u modulu Servlet verziji 2.2. U to vrijeme JavaScript i XML su već bili razvijeni ali se nisu koristili kao što se to danas koristi pomoću AJAX-a [3].

2005. godine princip AJAX-a se kreirao i aplikacije kao što je Gmail počele ga je koristiti i omogućila je klijentsku interakciju. Skripta koja se vezala uz samu web stranicu je mogla kontaktirati web server i pohraniti ili preuzeti podatke bez da se trebala preuzeti cijela web stranica [3].

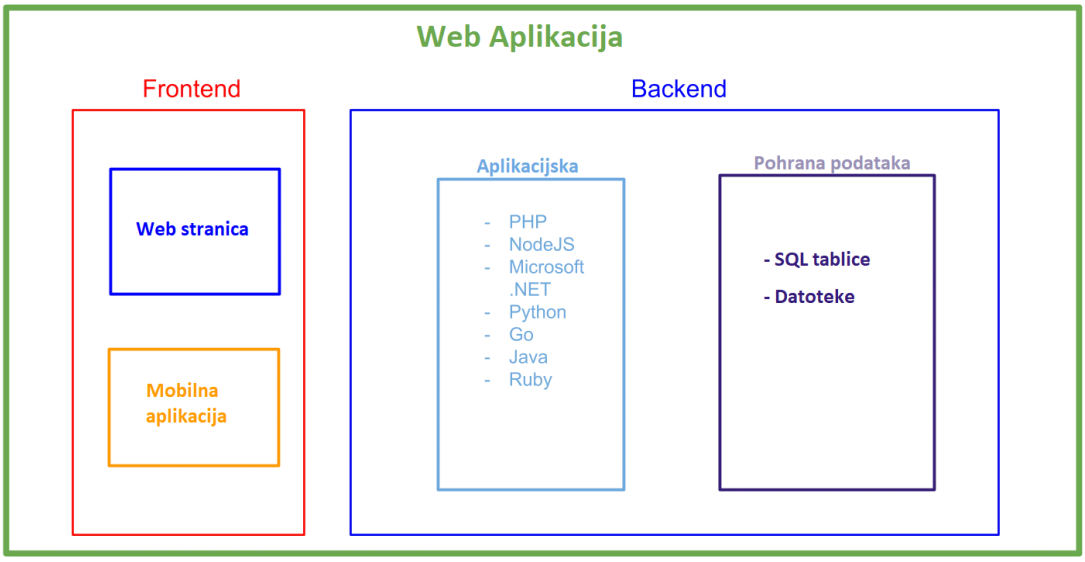
2014. godine HTML 5 je finaliziran koji je omogućio grafičko i multimedijsko korištenje bez plugin-ova s klijentske strane, te su zatim API i DOM su postali osnovni dijelovi HTML 5 specifikacije. Pojavio se i WebGL API za korištenje naprednih 3D grafike, koji je nastao bazirajući se na HTML 5 Canvas elementu i JavaScript-u [3].

4.2. Struktura web aplikacija

Sama struktura web aplikacija se razdjeljuje na logične dijelove ili redove, svaki dio ima posebnu ulogu koju izvodi. Tradicionalna struktura aplikacija se sastoji od jednog dijela koji se postavlja na klijentsko računalo, ali sama web aplikacija teži prema više zasebnih dijelova ili ti n-dijelova same aplikacije [2]. Najpoznatija i najkorištenija struktura web aplikacije sastoji se od 3 dijela:

- Prezentacijska
- Aplikacijska
- Pohrana podataka

Sami prezentacijski dio predstavlja web preglednik kojim se krajnji korisnik koristi, to može biti: „Google Chrome, Mozzila Firefox, Opera, Internet Explorer“. Aplikacijski dio predstavlja dinamički Web pokretač kao što su: „PHP, Node.js, Python, Ruby on Rails, ASP, CGI, Dart, JSP/Java, ColdFusion“, koji nam služi za neprestanu interakciju s web stranicom. Treći dio ili razina nam služi za pohranu svih podataka o korisniku ili podataka koji se prikazuju samom krajnjem korisniku, a ti podaci su najčešće pohranjeni u nekoj bazi podataka kojoj pristupamo pomoću Web servisa.



Slika 2. Struktura Web stranice [6]

5. Web servisi

Web servisi su modularne, samostalne, samo opisujuće aplikacije kojima se može pristupiti preko interneta, one su najpopularnije realizacije servisno-orijentirane arhitekture [4]. Oni sadrže kolekcije otvorenih protokola i standarda koji se koriste za razmjenu informacije između aplikacija ili sustava. Sama tehnologija je smještena na web serveru i može biti napisana u različitim programskim jezicima. Omogućava nam komunikaciju i razmjenu podataka između aplikacija preko internetske mreže, a podaci se prenose preko otvorenih protokola kao što su: TCP/IP, HTTP i koriste standarde formate kao što su: XML i JSON.

5.1. XML

Proširivi jezik za označavanje ili XML (eng. Extensible Markup Language) je jezik označavanja koji definira pravila kreiranja dokumenata koji su ujedno čitljivi korisniku, a tako i računalu ili aplikaciji. Njegov dizajn naginje jednostavnosti i fleksibilnosti za korištenje preko interneta. Nastao 1996. godine kao rješenje nedostataka od SGML-a koji je bio prekompleksan za korištenje, a i radi HTML-a koji je imao previše ograničenja [12].

Od početka kreiranja XML-a morao je zadovoljavati 10 točki [5]:

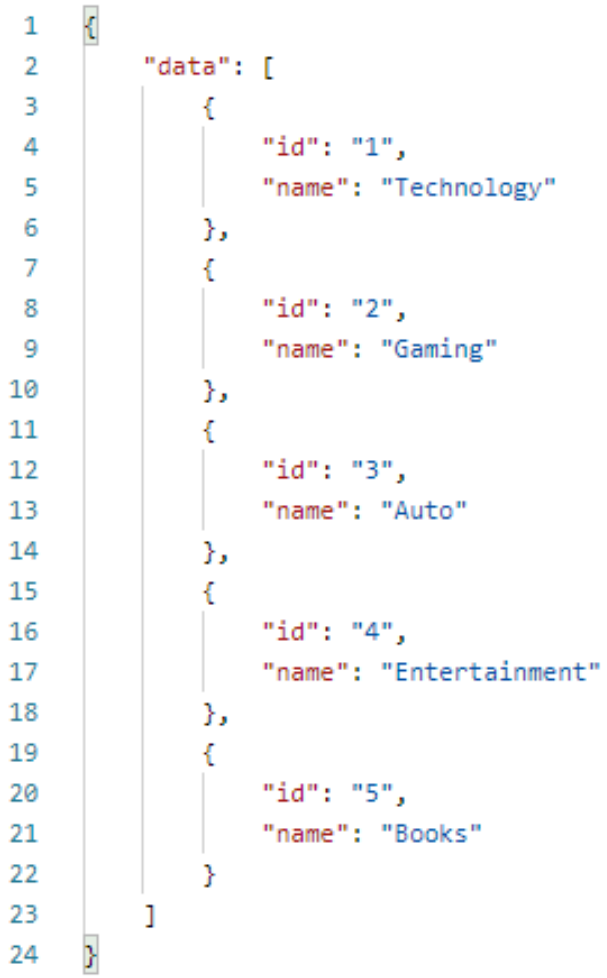
1. Mora biti jednostavan za korištenje preko interneta
2. Mora podržavati veliku raznolikost kod aplikacija
3. Mora biti kompatibilan sa SGML-om
4. Programi koji će procesuirati XML moraju biti jednostavni za pisanje
5. Opcionalni dijelovi koji se mogu uključiti u samu XML datoteku moraju biti svedeni na minimum, poželjno je da se radi o nula takvih slučajeva
6. XML dokument treba biti većinski lako čitljiv
7. XML-ov dizajn se mora lako kreirati
8. XML-ov dizajn treba biti formalan i sažet
9. XML dokument treba biti jednostavan kod kreiranja
10. Sažetost označavanja kod XML dokumenta je od minimalne važnosti

```xml
<?xml version="1.0" encoding="UTF-8"?>
<kategorije>
    <kategorija>
        <id>1</id>
        <name>Technology</name>
    </kategorija>
    <kategorija>
        <id>2</id>
        <name>Gaming</name>
    </kategorija>
    <kategorija>
        <id>3</id>
        <name>Auto</name>
    </kategorija>
    <kategorija>
        <id>4</id>
        <name>Entertainment</name>
    </kategorija>
    <kategorija>
        <id>5</id>
        <name>Books</name>
    </kategorija>
</kategorije>
```

Slika 3. XML format podataka

5.2. **JSON**

JavaScript objekt oznaka ili JSON (eng. JavaScript Object Notation) je otvoreni standard za formatiranje podataka kod prijenosa između aplikacija i ima ljudski čitljivu sintaksu. Podatke sprema u obliku atribut-vrijednost, a također može imati i polje kao tip podataka [13]. Kreirao ga je Douglas Crockford početkom 2000-te godine, ali je standardiziran tek 2013. godine. Treba napomenuti kako JSON nije ovisan o programskom jeziku, proizašao je iz JavaScript-a i mnogi ga programski jezici podržavaju. Tipovi podataka koje JSON format podržava su: „Integer, String, Boolean, Polja, Objekti, null“.



Slika 4. JSON format podataka

Kod web servisa od velike važnosti su i protokoli koji se koriste jer nam oni govore kako se podaci prenose i kako aplikacija komunicira preko internetske mreže. Neki od važnijih i popularnijih protokola su:

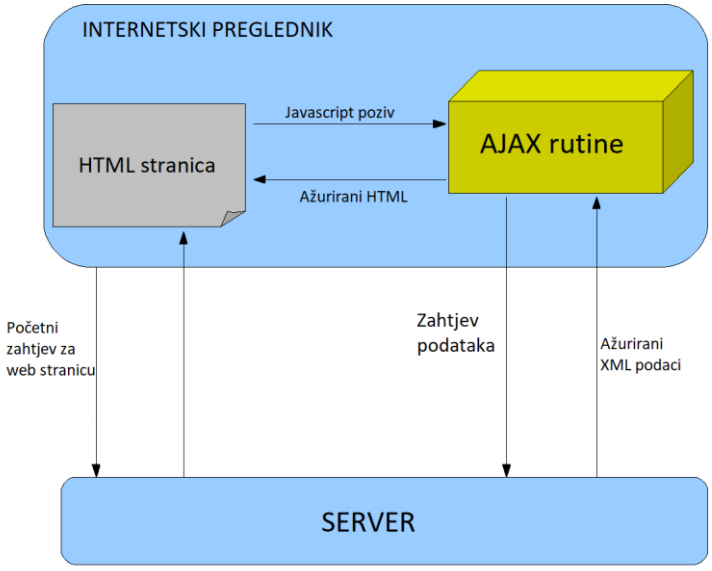
- AJAX
- SOAP
- REST

5.3. AJAX

AJAX (eng. Asynchronous JavaScript and XML) je kratica za „asinkroni JavaScript i XML“, a ono je web tehnika koja koristi mnoštvo web tehnologija na klijentskoj strani kako bi kreirao asinkronu komunikaciju kod korištenja web aplikacija. Sa AJAX-om podaci se mogu slati u pozadini prema web serveru bez nametanja samog prikaza web stranice. Takva komunikacija je moguća ako se prezentacijski i aplikacijski dijelovi web stranice odvoje, time se dobiva dinamičnost web stranice koju nije potrebno ponovno učitati nego se sam prikaz podataka izmjenjuje. Kod modernih implementacija AJAX-a više se koristi format prijenosa podataka JSON nego XML. Popularizira je tek 2005. godine od strane Google-a koji ga je koristio za „Google suggest“ ili autocomplete funkciju.

Tipični AJAX poziv bi se sastojao od: [10]

1. Klijent pokreće neki događaj sa pritiskom na gumb
2. Prisluškivač događaja-ova šalje HTTP zahtjev prema serveru
3. Server vraća rezultat zahtjeva u obliku XML/JSON
4. Web stranica se ažurira koristeći rezultate/podatke koje smo dobili od servera



Slika 5. AJAX protokol [21]

5.4. **SOAP**

SOAP (eng. Simple Object Access Protocol) ili „Jednostavni protokol za pristup objektima“ to je protokol poruka koji imaju unaprijed određeni format pomoću kojeg servis i aplikacija lakše komuniciraju. Postoji različiti tipovi poruka u SOAP-u, a najpoznatiji je poziv udaljenih procedura ili RPC (eng .Remote Procedure Call). Protokol se najbolje snalazi u okuženju gdje postoji formalni dogovor komunikacije između aplikacije i servisa, a tako i kod operacija gdje se pazi na stanje servisa ili aplikacije [11]. Velika mana SOAP servisa je prevelika kompleksnost i opširnost pri korištenju XML-a i trošenje resursa kako bi se ispravno parsirao XML.

Jednostavna XML poruka se sastoji od:

1. Omotač
2. Zaglavlje poruke
3. Tijelo poruke
4. Element stanja

Slika 6. SOAP protokol [22]

Omotač element je obavezni dio i služi nam kako bi mogli identificirati kako se radi o SOAP poruci koja sadrži XML i kako se ona parsira. Zaglavlje je opcionalni dio ali može sadržavati atribute koji se koriste kod procesiranja poruke. Tijelo je također obavezni dio SOAP poruke i

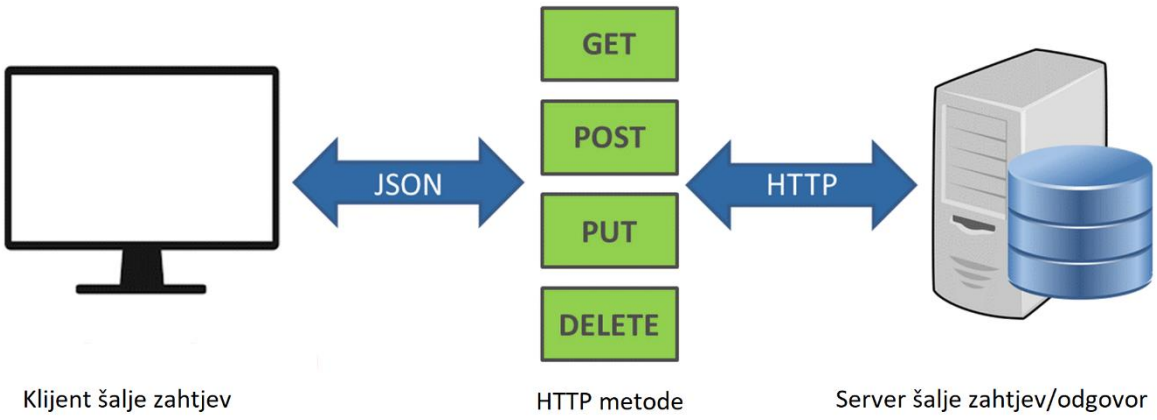
on prenosi informacije o dohvaćanju podataka ili može prenositi rezultate dohvaćanja. Element stanja je opcionalni dio i on nam pruža informacije što treba napraviti ako dođe do pogreške prilikom kreiranja same poruke.

5.5. **REST servisi**

REST (eng. Representational state transfer) ili „Reprezentativni prijenos stanja“ je softverska arhitektura koja definira unaprijed određena pravila i ograničenja koja moraju biti zadovoljena kod kreiranja Web servisa [13]. Takvi servisi omogućuju aplikacijama da pristupe i manipuliraju reprezentacijom sadržaja na web stranici. Kod RESTful web servisa upiti koji se ostvaruju preko URI dobiti će odgovor formatiran u jednom od sljedećih oblika: „HTML, JSON ili XML“, također odgovori mogu biti samo i reprezentacija stanja aplikacije ili podataka. Kako koristimo HTTP za slanje upita i odgovora, postoje i operacije koje su nam dostupne:

- GET
- HEAD
- POST
- PUT
- PATCH
- DELETE
- CONNECT
- OPTIONS
- TRACE

Kako se upiti šalju bez otvaranje sesije prema web serveru RESTful aplikacije ciljaju na performanse, pouzdanost i mogućnost da aplikacija skalira koristeći komponente koje se mogu nadograditi bez da promjena utječe na cijeli sustav.



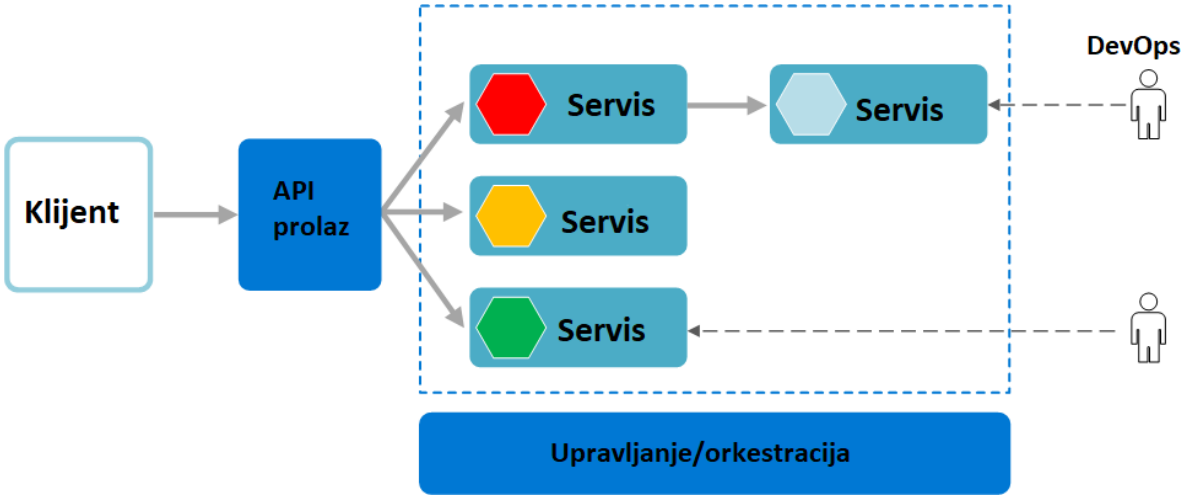
Slika 7. RESTful API [23]

6. Mikroservisi

Mikroservis je zapravo arhitektura koji se sastoji od kolekcije servisa koji su lako održivi i testivi, slabo međusobno povezani, zasebno razvijeni, organizirani oko samog biznisa i njegove potrebe [15]. Oni nam omogućavaju brzo, često i pouzdano razvijanje kompleksnih aplikacija, a tako i bolje održavanje tehnologija s kojima se aplikacija kreira. Na mikro servis ne treba gledati kao dio aplikacije nego to je zasebni dio funkcionalnosti biznisa koji nam služi kao sučelje za komunikacije između više servisa i/ili aplikacija. Aplikacije koje su prisvojile takvu arhitekturu su cloud-native, bez servera ili aplikacije koje koriste „lagane“ kontejnere za razvijanje same.

Beneficije takve arhitekture su:

- **Modularnost**
- **Skalabilnost**
- **Integracija**
- **Distribucijski razvoj**



Slika 8. Mikroservisi arhitektura [24]

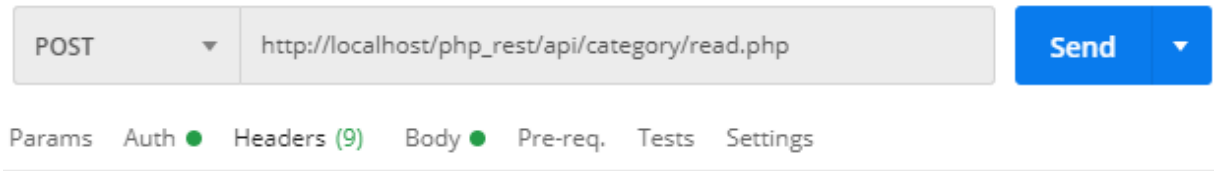
7. Web API

Web API (eng. Application programing interface) ili „aplikacijsko programsko sučelje“ je komplet definicija, protokola i alata koji nam služe za kreiranje software-a i aplikacija [16]. Ono se prvi put definiralo 1968. godine kao „kolekcija rutina koje dostavljaju klijentu željene podatke i njihove funkcionalnosti“ (Cotton and Greatorex, 1968).

Samim korištenjem i razvijanjem API-ja omogućava programerima štednju vremena i truda koji je potreban da se napiše neki složeni program, pri čemu svi programeri koriste iste standarde. Napretkom u operacijskim sustavima, osobito napretkom u grafičkom korisničkom sučelju, API je postao nezaobilazan element u stvaranju novih aplikacija. Umjesto da se novi programi pišu iz temelja, programerima se omogućuje nastavljanje na radu drugih. Njegova implementacija sučelja se nalaze na serverskoj strani ili backend-u.

7.1. Poslužiteljska strana API-a

Poslužiteljska strana API-a je programsko sučelje koji se sastoji od različitih krajnjih točaka ili „endpoint-ova“ koji reprezentiraju upite prema samom API-u. Krajnje točke imaju važnu ulogu jer reprezentiraju kojim resursima se može pristupiti. Standardna praksa kontaktiranja krajnjih točaka se ostvaruje pomoću URI (eng. uniform resource identifier) ili „jedinstveni identifikator resursa“ koristeći HTTP zahtjeve za kontaktiranje servera. Krajnje točke moraju biti statične kako bi omogućile stalnu konekciju prema serveru.
Upiti se ostvaruju pomoću linka i oni se unaprijed određuju kako bi se znalo kojim podacima se može pristupiti.



Slika 9. Upit prema Web API-u

Kod svakog slanja upita prema serveru dobiti ćemo HTTP odgovor od servera u kojem se nalazi statusni kod s kojim se zatim može provjeriti stanje odgovora.

7.2. Statusni kodovi

Statusni kodovi nam prikazuju stanje zahtjeva kojeg smo poslali s klijentske strane prema serveru. Postoje pet kategorija statusnih kodova koji su obavezni u znanju kod kreiranja same Web aplikacije [18]:

1. 1xx: Informacijski statusni kod
2. 2xx: Statusni kod uspjeha
3. 3xx: Statusni kod preusmjeravanja
4. 4xx: Statusni kod klijentske pogreške
5. 5xx: Statusni kod serverske pogreške

U slijedećem poglavlju proći ćemo detaljnije u same statusne kodove HTTP serverskog odgovora.

7.2.1. Informacijski statusni kodovi

Statusni kodovi koji započinju s brojkom „1xx“ su informativnog tipa i oni nam pružaju informacije o privremenom stanju odgovora, a oni se sastoje samo od statusne linije i opcionalnog zaglavlja poruke. Važna činjenica kod ovih kodova je da klijentska strana aplikacije mora biti spremna prihvatiti jedan ili više informacijskih statusnih kodova prije samog dobivanja odgovora, čak i ako ne očekuje statusnu poruku [18].

Statusni kodovi:

- **100 (Nastavak)** – Ovaj statusni kod govori klijentu kako je inicijalni dio zahtjeva je prihvaćen i trenutno nije odbijen od strane servera.
- **101 (Promjena protokola)** – Statusni kod koji obavještava klijenta o promjeni protokola koji se koristi i sve potrebne informacije se nalaze u zaglavlju odgovora.
- **102 (Obrada zahtjeva)** – Govori kako je server zaprimio zahtjev od klijenta i trenutno ga obrađuje i odgovor trenutno nije moguć.
- **103 (Rani nagovještaj)** – Statusni kod obavještava korisnika kako će podaci uskoro stići i da može krenuti s pripremanjem resursa prije nego što primi od servera finalni odgovor.

7.2.2. Statusni kodovi uspjeha

Ovi statusni kodovi nam govore o tome kako je klijentski zahtjev zaprimljen, shvaćen i prihvaćen od strane servera [18].

Statusni kodovi:

- **200 (Uspjeh)** – Zahtjev je uspješan i podaci su se dostavili do klijenta preko jedne od metoda koje se koriste.
- **201 (Kreiran)** – Govori kako je zahtjev bio uspješan i novi resurs je kreiran kao rezultat zahtjeva.
- **202 (Prihvaćen)** – Zahtjev je primljen na obrađivanje, ali samo procesuiranje zahtjeva još nije gotovo. Ovaj statusni kod se većinom koristi kod dnevnika i kod velike količine podataka.
- **203 (Neautoritativne informacije)** – Ovaj statusni kod nas obavješćuje kako meta informacije u zaglavlju nisu iz originalnog servera nego su preuzete iz lokalne ili treće strane.
- **204 (Nema podataka)** – Server vraća odgovor kako je zahtjev obrađen ali ne treba vratiti podatke u odgovoru. Server u ovom slučaju može vratiti samo ažurirane meta informacije.
- **205 (Resetiranje podataka)** – Govori klijentu kako bi trebao resetirati dokument koji šalje zahtjev.
- **206 (Polovične podataka)** – Statusni kod se koristi kod svakog zahtjeva koji ima varijablu „raspon“ u zaglavlju i zahtjeva samo jedan dio podataka.

7.2.3. Statusni kod preusmjeravanja

Statusni kodovi 3xx nam govore kako su potrebne dodatne akcije kako bi zahtjev se mogao dovršiti, u nekim slučajevima klijentska strana će sama nadodati potrebne informacije kako bi se zahtjev mogao završiti, a u drugim slučajevima potrebna će biti interakcija sa samim korisnikom. Moguće je i da sami zahtjev je kriv i konstantno se vrti u kruh koji potom stvara nepotrebno mrežno opterećenje, u tom slučaju korisnička strana treba prepoznati ponavljanje takvih zahtjeva i obustaviti ih [18].

Statusni kodovi:

- **300 (Više izbora)** – Željeni zahtjev ima više mogućih odgovora, klijentska strana je tada obavezna odabrati jedan od odgovora.
- **301 (Zauvijek pomaknuto)** – Govori kako je URL koji se koristio za dohvaćanje podatka promijenjen za stalno. U zaglavlju odgovora se nalazi novi URL koji se koristi za dohvaćanje željenih informacija u varijabli „Location“.
- **302 (Pronađeno)** – Govori kako je URL koji se koristio za dohvaćanje podatka promijenjen privremeno, a u zaglavlju odgovora se nalazi novi URL koji se koristi za dohvaćanje željenih informacija u varijabli „Location“.
- **303 (Pogledaj druge)** – Odgovor se nalazi pod drugim URI-em i trebao bi se dohvatiti koristeći GET metodu na tom resursu.
- **304 (Nije modificirano)** – Govori klijentu kako odgovor nije bio dodatno modificiran.
- **305 (Koristi proxy)** – Kako bi se pristupilo odgovoru na zahtjev treba to učiniti preko proxy-a.
- **306 (Nije u upotrebi)** – To je rezerviran statusni kod od prijašnje specifikacije i nije u upotrebi.
- **307 (Privremeno preusmjeravanje)** – Statusni kod nas obavještava kako se zahtjev mora zatražiti preko novog URI-a ali se koristi ista metoda. Ovaj statusni kod je sličan „302“ statusnom kodu ali je naglašeno kako se koristi ista HTTP metoda.
- **308 (Stalno preusmjeravanje)** – Statusni kod nas obavještava kako se lokacija promijenila za stalno na nekom novom URI-u, taj novi link za pristup se nalazi u zaglavlju samog odgovora u varijabli „Location“. Sličan je „301“ statusnom kodu sa iznimkom da će se koristiti ista HTTP metoda.

7.2.4. Statusni kodovi klijentske pogreške

Statusne kodove „4xx“ nam govore kako se desila pogreška kod slanja zahtjeva na klijentskoj strani. Osim u slučaju kada se odgovora na HEAD zahtjev u tom slučaju server mora obavijestiti korisnika o pogrešci koja se dogodila. U ovom poglavlju neće se pokriti svi statusni kodovi tipa „4xx“ nego samo najvažniji [18].

Statusni kodovi:

- **400 (Loš zahtjev)** – Sintaksa zahtjeva nije bila razumljiva i ne može se obraditi od strane servera.
- **401 (Neovlašteno)** – Obavješćuje kako obrada zahtjeva je moguća uz ispravnu korisničku autentikaciju.
- **402 (Zahtjeva uplatu)** – Koristi se kod digitalnih uplata.
- **403 (Zabranjeno)** – Neovlašteni pristup, klijent nema prava pristupu podacima.
- **404 (Nije pronađeno)** – Server ne može pristupiti podacima koje zahtjev traži.
- **405 (Metoda nije dozvoljena)** – Obavještava kako HTTP metoda je poznata serveru ali je blokirana.
- **406 (Nije prihvaćeno)** – Server ne može pronaći podatke koji se traže od strane korisnika preko „Accept“ varijable u zaglavlju zahtjeva.
- **407 (Autentikacija preko proxy-a potrebna)** – Obavještava klijenta kako se prvo mora identificirati preko proxy-a kako bi mogao pristupiti podacima.
- **408 (Zahtjev istekao)** – Govori kako server nije primio potpuni zahtjev od klijenta u vrijeme koje je određeno na serveru za primanje zahtjeva.
- **409 (Konflikt)** – Zahtjev nije bio moguće obraditi jer je došlo do konflikta s trenutnim stanjem resursa.
- **410 (Nestao)** – Podaci koji se traže u zahtjevu više ne postoje na serverskoj strani

7.2.5. Statusni kodovi serverske pogreške

Statusni kodovi serverske pogreške nas obavještavaju da je došlo do nepredvidive situacije na serverskoj strani koja blokira obradu zahtjeva [18].

Statusni kodovi:

- **500 (Unutarnji problemi)** – Server je naišao na probleme koji blokiraju obradu zahtjeva.
- **501 (Nije implementirano)** – HTTP metoda nije implementirana na serveru i zahtjev ne može biti obrađen.
- **502 (Loš poveznik)** – Server je zaprimio loš odgovor pri ulozi poveznika i nije u mogućnosti obraditi zahtjev.
- **503 (Servis nedostupan)** – Server nije spreman obraditi zahtjev.
- **504 (Isteklo vrijeme poveznika)** – Server je u ulozi poveznika i nije na vrijeme dobio odgovor i nije u mogućnosti obraditi zahtjev.
- **505 (HTTP verzija nije podržana)** – Govori nam kako HTTP verzija korištena u zahtjevu nije podržana od strane servera.
- **510 (Nije očekivano)** – Obavještava nas kako su potrebne dodatne informacije o zahtjevu kako bi server mogao ispuniti zahtjev.
- **511 (Mrežna autorizacija potrebna)** – Govori kako klijent se treba identificirati kako bi dobio pristup mreži.

8. Važnost dokumentacije

Tehnička dokumentacija predstavlja jedinstveni jezik komunikacije između razvijatelja programske potpore i ostalih stručnjaka koji imaju interes za projektom. Iz tog razloga ona čini jedan od najvažnijih oblika dokumenata programske potpore, a dobru dokumentaciju čini njezina jednostavnost, sažetost i lakoća razumijevanja [19]. Neke od smjernica koje bi trebalo pratiti kod pisanja dokumentacije su:

- Razumijevanje kome je dokumentacija namijenjena
- Kratki ali detaljni opisi koji ciljaju glavne elemente aplikacije
- Prikazati dijelove koda kako bi čitatelj imao bolje razumijevanje pojedinačnih dijelova aplikacije
- Konstantno ažuriranje dokumentacije pri svakoj većoj promjeni u aplikaciji
- Kod opširnih projekata od velike je važnosti da sudionici projekta prate ove smjernice kako bi uvelike smanjiti vrijeme rješavanja problema u budućnosti ili omogućili novim programerima lakše priključivanje projektu i razumijevanje same aplikacije koja se nadograđuje.

9. Razlike SOAP API-a i REST API-a

Prije nego što započnemo s tehničkim dijelom rada moramo razumjeti što ćemo kreirati i potom prikazati. Kada uspoređujemo sami SOAP API i RESTful API, treba zamijetiti da svaki Web API nije RESTful, ali zato svaki RESTful je Web API. To znači da sam Web API je programsko sučelje koje nam omogućava pristup informacijama/podacima koji su nam potrebni pri radu Web aplikacije. A RESTful API je komplet pravila, standarda i smjernica koje možemo smatrati kao Web arhitekturu koja nas upućuju kako izgraditi klijentsku i serversku stranu Web aplikacije. U praktičnom dijelu kreirati ću komunikaciju između dvoje Web API-a, njihovim pozivima biti će prikazana razlika između SOAP API-a i RESTful API-a.

Elementi Web API-a:

- Koristi HTTP protokol
- Podaci prikazani u XML formatu

Elementi RESTful API-a:

- Koristi HTTPS protokol
- Podaci prikazani u JSON formatu
- Bolji nazivi elemenata, a tako i razumijevanje linkova kojima se šalje zahtjev
- Dobra pokrivenost statusnih kodova

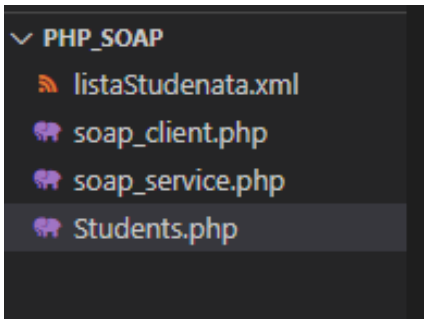
10. Opis praktičnog dijela rada

Tehnički dio završnog rada se odnosi na predstavljanje problema, a tako i komunikacije između različitih API-a i prikazivanje njihovih rješenja. Cijeli interface je pisan u PHP-u, Javascript-u, HTML, CSS. Krenuti čemo od same konstrukcije programskog koda i kasnije pojedinačno prikazati interoperabilnost servisa.

10.1. SOAP API

SOAP servis se sastoji od troje važnih datoteka koji ćini jednu cijelinu**,** na slici 10. možemo vidjeti cijeli datotečni sustav sučelja SOAP servisa:

- **Soap_client.php** – klasa koja nam služi kao klijent ili preko koje korisnik može pristupiti servisu
- **Soap_service.php** – klasa koja na služi kao server preko kojeg budem dobivali sve podatke, a također služi za obradu pogrešaka koje se mogu stvoriti tokom obrade zahtjeva
- **Students.php** – klasa koja nam služi za pohranu i dobavljanje podataka koji se mogu dohvatiti preko zahtjeva na SOAP servis



Slika 10. Datoteke SOAP servisa

Krenimo od osnovnih dijelova samog API-a, a to se prvenstveno odnosi na podatke koji se mogu dohvatiti preko servisa, njih smo spremili u datoteku ili klasu pod nazivom Students.php u višedimenzionalno polje. U praksi podaci nisu spremljeni na ovakav način nego postoje baze podataka koje su povezane sa servisom.

```
<?php
class Students
{
    public function getStudents(){
        $studenti = array( array("Mateo", "Matic", "26"),
                           array("Iva", "Matic", "45"),
                           array("Stjepan", "Matic", "49"),
                           array("Mihael", "Matic", "19"));
        return $studenti;
    }
}
?>
```

Sljedeći dio koda se odnosi na „soap_service.php“ ili serversku stranu SOPA servisa koji nam služi kao medij između klijentske strane i baze podataka. Ova klasa nam omogućuje odabir baze podataka iz koje dobavljamo podatke, a također nam služi za obradu pogrešaka koje se mogu stvoriti prilikom obrade podataka, to čini funkcija „handle()“.

```
<?php
require_once('Students.php');

$option = array("uri" => "http://localhost");

$server = new SoapServer(null, $option);

$server->setClass('Students');
$server->handle();

?>
```

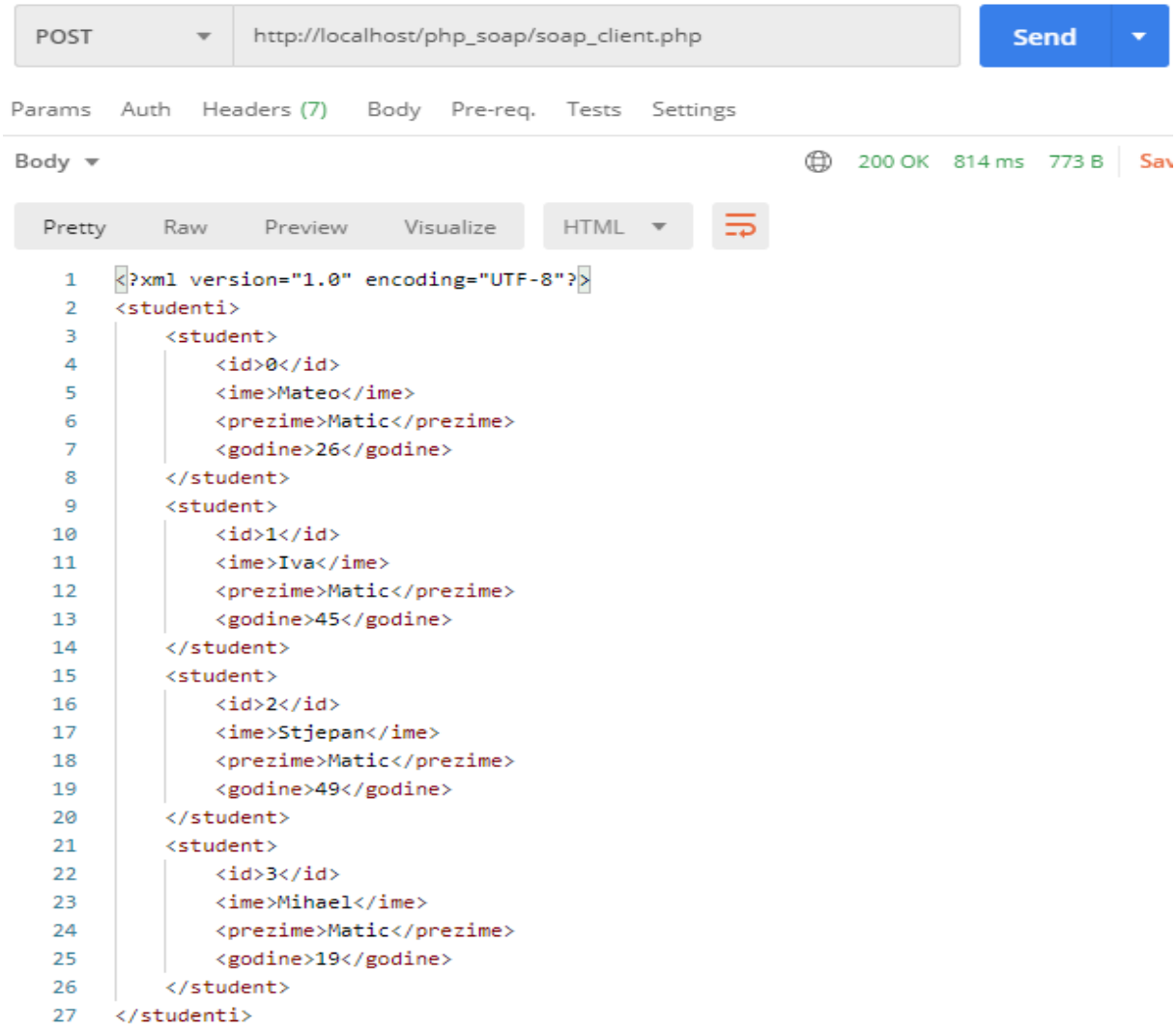
Zadnji dio SOAP servisa se odnosi na klasu „soap_client.php“ koji na služi kao prednja strana (eng. frontend) servisa. Krenuvši od početka koda prvenstveno se određuje URL ili „jedinstveni lokator resursa“ (eng. Uniform Resource Locator) koji nam služi kao krajnja točka za slanje zahtjeva prema SOAP servisu. Nakon definiranja URL-a možemo inicijalizirati naš SOAP klijent u kojem navodimo opcije koje želimo uključiti. Nakon ta dva koraka spremni smo dohvatiti podatke i krenuti ih obrađivati, podaci se dohvaćaju preko klase „Student.php“ s funkcijom „getStudents()“. Dohvaćene podatke zatim spremamo u XML format kako bi bili spremni za daljnje prosljeđivanje. Treba naglasiti da cijeli SOAP klijent treba biti postavljen u try-catch blok koda kako bi mogli prepoznati pogrešku u datom trenutku.

```
<?php

$option = array("location" => "http://localhost/php_soap/soap_servic
e.php", "uri" => "http://localhost", 'trace' => 1);

try{
    $client = new SoapClient(null, $option);

    $students = $client->getStudents();

    $xml = new DomDocument("1.0","UTF-8");
    $studenti = $xml->createElement("studenti");
    $studenti = $xml->appendChild($studenti);

    foreach($students as $i=>$b){

        $student = $xml->createElement("student");
        $id = $xml->createElement("id", $i);
        $ime = $xml->createElement("ime", $b[0]);
        $prezime = $xml->createElement("prezime", $b[1]);
        $godine = $xml->createElement("godine", $b[2]);

        foreach($b as $j=>$a){
            $student = $studenti->appendChild($student);

            $id = $student->appendChild($id);

            $ime = $student->appendChild($ime);

            $prezime = $student->appendChild($prezime);

            $godine = $student->appendChild($godine);
        }
    }

    $xml->formatOutput = true;
    $string_value = $xml->saveXML();
    $xml->save("listaStudenata.xml");

    $file = file_get_contents('./listaStudenata.xml');
    echo $file;

    //echo "\nREQUEST:\n" . htmlentities($client-
>__getLastRequest()) . "\n";
}catch(SoapFault $ex){
    var_dump($ex);
}
?>
```

Primjetimo li zakomentiranu liniju prije „catch“ bloka koda možemo vidjeti kako nam SOAP servis omogućuje pregled zadnjeg poslanog zahtjeva sa svim potrebnim elementima koje zahtjev mora imati prilikom slanja.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="http://localhost"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
     <ns1:getStudents/>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

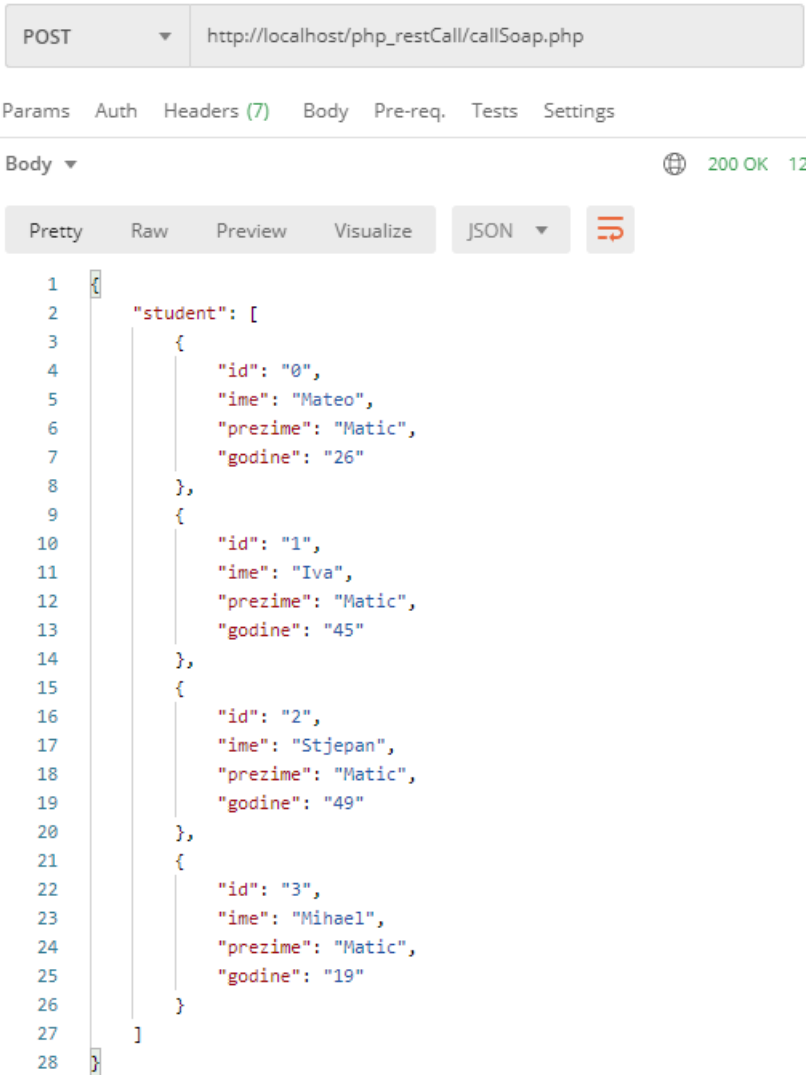
URL kojeg koristimo i preko kojeg možemo pristupiti SOAP servisu je http://localhost/php_soap/soap_client.php i njegov odgovor možemo vidjeti na slici 11.



Slika 11. Odgovor na zahtjeva prema SOAP servisu

10.2. **REST API**

Nakon prikaza SOAP API-a prelazimo na RESTful API koja je mnogo opširnije korišten u svijetu nego prijašnja verzija. Odma prva razlika koju možemo zamjetiti je korištenje JSON formata prijenosa podataka. Korištenje takvog formata nam daje veliku slobodu kod same pripreme podatka koje želimo dostaviti prema krajnjem korisniku.



Slika 12. Odgovor REST API-a

Iz slike 12. možemo primijetiti da ponovo budemo pozvali slični zahtjev o čitanju podataka ali samo prema novom REST API-u. Vidimo kako je odgovor dobiven u JSON formatu, spremljen pod jednu grupu podataka „student“ za lakše pristupanje, statusni kod je „200“ što znači da je upit uspješno ostvaren. Također možemo zamijetiti kako sami upit se ostvaruje preko „POST“ metode koja je sigurnija i današnji standard za dohvaćanje podataka.

Sami REST API se sastoji samo od:

- **callSoap.php** – klasa koja nam služi za interoperabilnost zajedno s SOAP servisom, a također i u njoj obrađujemo podatka koje smo zaprimili
- **indeks.html** – je jednostavna HTML stranica koja sadrži elemente CSS i JavaScript-a koji nam omogućuju jednostavni prikaz podataka u tablica koje smo dobili

Krenimo od osnovne datoteke „callSoap.php“ koja nam služi za obradu zahtjeva i pozivanja drugog servisa kako bi dobili potrebne podatke i ostvarili interoperabilnost između dva servisa. Na početku koda pozivamo funkciju koju smo kreirali kako bi ostvarili kontakt s SOAP servisom, samoj funkciji potrebna su tri preduvjeta, a to su:

- Metoda zahtjeva
- URL prema kojem šaljem zahtjev
- Podaci (eng. data) koji se mogu ili nemoraju slati prema drugom servisu

Svaki od ovih preduvjeta je ključan za uspješno slanje zahtjeva, sami zahtjev šaljemo preko „curl“ funkcije koja nam uvelike olakšava posao. Podaci koje zaprimamo od strane SOAP servisa su u obliku XML i treba ih pretvoriti u JSON standardi oblik za obradu podataka kod servisa. Također te podatke spremamo u zasebni JSON datoteku kako bi mogli dalje koristiti podatke koje smo zaprimili.

```
<?php

callApi("POST", "http://localhost/php_soap/soap_client.php", true);
function callAPI($method, $url, $data = false){
    $curl = curl_init();

    switch($method){
        case "POST":
            curl_setopt($curl, CURLOPT_POST, 1);
            if($data){
                curl_setopt($curl, CURLOPT_POSTFIELDS, $data);
            }
            break;
        case "PUT":
            curl_setopt($curl, CURLOPT_PUT, 1);
```

```
            break;
        default:
        if ($data)
            $url = sprintf("%s?%s", $url, http_build_query($data));
    }

    curl_setopt($curl, CURLOPT_URL, $url);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);

    $result = curl_exec($curl);

    curl_close($curl);

    $xml = simplexml_load_string($result, "SimpleXMLElement", LIBXML_NOCDAT
A);
    $json = json_encode($xml);

    $fp = fopen("result.json", "w");
    fwrite($fp, $json);
    fclose($fp);

    echo $json;
    return $json;
}
?>
```

Zatim prelazimo na „index.html“ stranicu koja nam služi za jednostavni prikaz zaprimljenih podataka, na njoj se nalazi i dva gumba koja na služe za slanje zahtjeva prema SOAP servisu, a drugi služi kako bi podatke mogli prikazati u tablici.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript" src="http://ajax.googleapis.com/ajax
/libs/jquery/1.6.2/jquery.min.js"> </script>
        <script>
        function stvoriTable(){
            $(function() {
                $.getJSON('result.json')
                    .success(function(data) {
                        cache: false,
                        $('table tr').remove();
                        data = data.student;
                        console.log(data);
                        var tr;
                        for (var i = 0; i < data.length; i++) {
                            tr = $('<tr/>');
                            tr.append("<td>" + data[i].id + "</td>");
                            tr.append("<td>" + data[i].ime + "</td>");
                            tr.append("<td>" + data[i].prezime + "</td>")
                            tr.append("<td>" + data[i].godine + "</td>")
                            $('table').append(tr);
                        }
                    })
                .error(function(e) { console.error(e); })
            });
        }
        function pozivRest(){
            $('.poziv').click(function() {
                $.ajax({
                    type: "POST",
                    url: "callSoap.php"
                }).done(function( msg ) {
                    alert( "Data Recieved: " + msg );
                });
            });
        }
        </script>
        <style>
            table, th, td {
                border: 1px solid black;
```

```
                text-align: center;
                width: 300px;
                margin-top: 30px;
            }
            table.center{
                margin-right: auto;
                margin-left: auto;
            }
        </style>
    </head>
    <body style = "text-align:center;" id = "body">
        <button class="poziv" onclick="pozivRest()">SEND REQUEST</button>
        <button onclick="stvoriTable()">CREATE TABLE</button>
        <table class="center">
            <tr>
                <th> ID </th>
                <th> Ime </th>
                <th> Prezime </th>
                <th> Godine </th>
            </tr>
        </table>
    </body>
</html>
```

11. Zaključak

Web aplikacije nam omogućuju razmjenu različitih informacija između sustava putem interneta, a da bi se to moglo uspješno napraviti potrebna nam je dobra arhitektura samog Web API-a, a tako i same aplikacije ili sustava. Neke od osnovnih karakteristika Web aplikacija su skalabilnost, interakcije između servisa, nezavisnost komponenti i predefinirani formati podataka.

REST API nam pruža takve karakteristike i još pritom je jednostavan za implementaciju i održavanje takvog sustava. Kod kreiranja REST API-a treba ga upariti koristeći HTTP protokol koji pruža REST arhitekturi jednoliko sučelje i privremeno pohranjivanje podataka u memoriju.

Osnovni cilj ovog rada je bio prikazati interoperabilnost između običnog SOAP API-a koji koristi zastarjele formate strukture podataka i REST API-a koji koristi novije metode a tako i formate podataka, koji nam olakšavaju baratanje njima samima.

12. Popis literature

1) Vince Bruno, Audrey Tam, James Thom, „Characteristics of web applications that affect usability“, 2005 [Na internetu]. Dostupno: https://www.academia.edu/16585608/Characteristics_of_Web_applications_that_affect_usability_A_Review

2) Swapnil Banga, „What is Web Application Architecture? Components, Models and Types“, 2020 [Na internetu]. Dostupno: https://hackr.io/blog/web-application-architecture-definition-models-types-and-more

3) Oleg Uryutin, „A brief history of web app“, 2018 [Na internetu]. Dostupno: https://medium.com/@aplextor/a-brief-history-of-web-app-50d188f30d

4) Cary Pennington, Jorge Cardoso, John A. Miller, Richard Scott Patterson, Ivan Vasquez, 2006, [Na internetu]. Dostupno: https://www.researchgate.net/publication/236860265_Introduction_to_Web_Services

5) Laura Papaleo „Introduciton to XML and its applications“, 2013 [Na internetu]. Dostupno: https://www.researchgate.net/publication/304823432_Introduction_to_XML_and_its_applications

6) Caleb H. „Web application stuctrure from web developers point of view“, 2019 [Na internetu] Dostupno: https://naclcaleb.hashnode.dev/web-application-structure-from-a-web-developers-point-of-view-cjxg46ljk001z01s1xvahvm3x

7) Saikou Y. Diallo, Heber Herencia-Zapana, Jose J. Padilla, Andreas Tolk, „Understanding Interoperability“, 2011 [Na internetu]. Dotupno: https://www.researchgate.net/publication/220954268_Understanding_interoperability

8) Chito Jovellanos, „Semantic and Syntactic Interoperability in Transactional Systems“, 2003 [Na internetu]. Dostupno: https://www.researchgate.net/publication/221444883_Semantic_and_syntactic_interoperability_in_transactional_systems

9) Marcia Lei Zeng, „Interoperability“ [Na internetu]. Dostupno: https://www.isko.org/cyclo/interoperability#3

10) Pat Morin, „Introduction to AJAX“ [Na internetu]. Dostupno: http://cglab.ca/~morin/teaching/2405/notes/ajax1.pdf

11) Margaret Rouse, „SOAP (Simple Object Access Protocol)“ [Na internetu]. Dostupno: https://searchapparchitecture.techtarget.com/definition/SOAP-Simple-Object-Access-Protocol

12) Laura Papaleo, „Introduction to XML and its Applications“, 2013 [Na internetu]. Dostupno: https://www.researchgate.net/publication/304823432_Introduction_to_XML_and_its_applications

13) Franklin Intriago, „JSON Tutorial“, [Na internetu]. Dostupno: https://www.academia.edu/19435559/JSON_Tutorial

14) Ralph Johnson, „REST and Web Services: In Theory and in Practice“, 2011 [Na internetu]. Dostupno: https://www.researchgate.net/publication/265236489_REST_and_Web_Services_In_Theory_and_in_Practice

15) https://www.researchgate.net/publication/326945322_Application_of_microservice_architecture_in_cloud_environment_project_development [Mikroservisi]

16) Ling Zheng, Bo Wei, „Application of microservice architecture in cloud environment project development“, 2018 [Na internetu]. Dostupno: https://cloud.google.com/files/apigee/apigee-web-api-design-the-missing-link-ebook.pdf

17) „HTPP Status Codes“, [Na internetu]. Dostupno: https://restfulapi.net/http-status-codes/

18) „10 status COde Definitions“, [Na internetu]. Dostupno: https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

19) „Dokumentiranje programskih rješenja“, 2011 [Na internetu]. Dostupno: https://www.cis.hr/files/dokumenti/CIS-DOC-2011-06-015.pdf

20) „Arhitektura Web Aplikacija“ [Na internetu] Dostupno: https://www.successcenter.com/upload/641126konstantthirdpartybanners41.png

21) „Create Ajax funtion“ , [Na internetu]. Dostupno: https://blog.arvixe.com/opencart-create-ajax-function/

22) Navneet R., „Understand SOAP Message Structure“ , 2018 [Na internetu]. Dostupno: https://codenuclear.com/understand-soap-message-structure/

23) Joseph Benharosh, „What is REST API? In plain English“, 2018 [Na internetu]. Dostupno: https://phpenthusiast.com/blog/what-is-rest-api

24) „Microservices arachitecture style“, 2019, [Na internetu]. Dostupno: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

13. Popis slika

14. Prilozi

1) Web_api.rar datoteka (u njoj se nalazi oba Web servisa sa datotekama navedenim u prijašnjih poglavljima, a one su potrebne za prikaz praktičnog dijela rada) [Na internetu] Dostupno: https://easyupload.io/jsp7ew