Razvoj Web API-ja za inventuru

Viduka, Ivan

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

Sveučilišni studij

IZRADA APLIKACIJSKO PROGRAMSKOG SUČELJA ZA INVENTURU

Završni rad

Ivan Viduka

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

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

Osijek, 09.07.2020.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Ivan Viduka
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4155, 16.09.2019.
OIB studenta:	48971408826
Mentor:	Izv. prof. dr. sc. Ivica Lukić
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Razvoj Web API-ja za inventuru
Znanstvena grana rada:	**Informacijski sustavi (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	09.07.2020.
Datum potvrde ocjene Odbora:	15.07.2020.

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

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 17.07.2020.

Ime i prezime studenta:	Ivan Viduka
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4155, 16.09.2019.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Razvoj Web API-ja za inventuru**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Ivica Lukić

i sumentora

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

Potpis studenta:

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

IZJAVA

o odobrenju za pohranu i objavu ocjenskog rada

kojom ja __Ivan Viduka_____, OIB: __48971408826_____, student/ica Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek na studiju _Preddiplomski sveučilišni studij Računarstvo_, kao autor/ica ocjenskog rada pod naslovom: _Razvoj Web API-ja za inventuru**,**
dajem odobrenje da se, bez naknade, trajno pohrani moj ocjenski rad u javno dostupnom digitalnom repozitoriju ustanove Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek i Sveučilišta te u javnoj internetskoj bazi radova Nacionalne i sveučilišne knjižnice u Zagrebu, sukladno obvezi iz odredbe članka 83. stavka 11. *Zakona o znanstvenoj djelatnosti i visokom obrazovanju* (NN 123/03, 198/03, 105/04, 174/04, 02/07, 46/07, 45/09, 63/11, 94/13, 139/13, 101/14, 60/15).
Potvrđujem da je za pohranu dostavljena završna verzija obranjenog i dovršenog ocjenskog rada. Ovom izjavom, kao autor/ica ocjenskog rada dajem odobrenje i da se moj ocjenski rad, bez naknade, trajno javno objavi i besplatno učini dostupnim:

a) široj javnosti
b) studentima/icama i djelatnicima/ama ustanove
c) široj javnosti, ali nakon proteka 6 / 12 / 24 mjeseci (zaokružite odgovarajući broj mjeseci).

**U slučaju potrebe dodatnog ograničavanja pristupa Vašem ocjenskom radu, podnosi se obrazloženi zahtjev nadležnom tijelu Ustanove.*

__Osijek, 17.07.2020._____ _____

(mjesto i datum) (vlastoručni potpis studenta/ice)

SADRŽAJ

1. UVOD

Razvojem tehnologije i provođenjem digitalizacije, znatno su ubrzani i olakšani svakodnevni poslovi. Spremanje velike količine informacija u razne baze podataka omogućava nam gotovo trenutni pristup na zahtjev te brze izmjene samih informacija. Također, omogućena je i interakcija s navedenim bazama podataka te njihovo korištenje u raznim programima. Kako bi se uštedilo na vremenu koje je potrebno za pisanje programskog koda, te kako bi se funkcije odrađivali na gotovo identičan način, sve je češća praksa korištenje aplikacijsko programskog sučelja. To sučelje omogućava razvojnim programerima pristup raznim metodama, za koje programer ne mora nužno znati kako rade, već samo mora poznavati potrebne ulazne parametre te što dobiva kao rezultat.

U ovom završnom radu prikazat će se jedno takvo sučelje koje olakšava obavljanje inventure na fakultetu. U kombinaciji s mobilnom aplikacijom, sučelje omogućava trenutan pristup informacijama o nekom objektu, putem jednostavnog skeniranja crtičnog koda.

Rad se sastoji od pet poglavlja. Nakon uvoda, drugo poglavlje opisuje aktualne praktične dosege (eng. *State of the Art*) vezane uz korištenje aplikacijsko programskog sučelja. Nadalje, treće poglavlje sadrži informacije o tehnologijama koje su korištene pri izradi ovog sučelja te koja je njihova uloga u testiranju ispravnosti programa. Četvrto poglavlje sadrži prikaz samog programskog koda te opis svih metoda koje se nalaze u sučelju, kao i komentiranje njihovih rezultata. Na kraju, zaključci se nalaze u petom poglavlju.

1.1. Zadatak završnog rada

Zadatak završnog rada je kreiranje aplikacijsko programskog sučelja, koje se povezuju s bazom podataka u kojoj su sadržani podaci za inventuru na fakultetu. Spremljene su informacije o svim prostorijama fakulteta te predmetima koji se u njima nalaze. Sučelje sadrži metode koje, ovisno o predanom parametru, dohvaćaju željene predmete, te metodu koja omogućava dodavanje nove stavke inventure u tablicu baze podataka.

2. PREGLED PODRUČJA TEME

Aplikacijsko programsko sučelje (skraćeno API) skup je rutina, protokola i alata koji se koriste u izgradnji programskih aplikacija. API definira kako različite programske komponente djeluju međusobno. Kvalitetno napravljeno sučelje omogućava jednostavnije i brže razvijanje programa. Nadalje, olakšava programerima dodavanje željene funkcionalnosti u aplikaciju, bez potrebe da sami pišu programski kod, te olakšava pristup podacima drugih aplikacija. U današnje vrijeme, aplikacijsko programska sučelja koriste se u gotovo svim kategorijama. Neke od njih su: vremenska prognoza, sportovi i profesionalne lige, glazba, vijesti, pretraživanja, web, API napravljen za interakciju s popularnim aplikacijama (Google Maps, Twitter, YouTube, …). Postoji brojna literatura u kojoj se postepeno objašnjava izrada sučelja, poput [1] i [2]. Nadalje, sve je više knjiga i radova koji su pisani specijalizirano za jednu vrstu API-ja ili jednu vrstu programskog jezika u kojem je pisan.

U ovom radu koristi se Web API [3][4], sučelje kojem se može pristupiti koristeći HTTP protokol. Iznimno je korisno jer nije potrebno posebno podešavati postavke za svaki uređaj koji koristi neku od HTTP usluga. Također, podržava rad s različitim oblicima podataka, kao što su XML ili JSON. Web API je moguće napisati u gotovo svim programskim jezicima, a neki češće korištenih su Java [5], JavaScript [6], Python [7] i C++ [8]. Za ovaj zadatak, korišten je programski jezik C# i .NET Framework. Od metoda, obrađena je POST te nekoliko varijanti GET funkcije. Ostale HTTP metode koje se znaju koristiti, poput PUT ili DELETE, nisu implementirane jer nisu tražene u zahtjevima za izradu API-ja. Međutim, implementacija tih metoda može se pronaći u ranije navedenoj literaturi.

U zadnjih nekoliko godina, uporaba Web API-ja doživjela je eksponencijalni rast. Osim u radu programera, sve se više koriste i kao marketinški alat u povećanju popularnosti proizvoda i usluga. Nadalje, povećava se i uporaba kod krajnjih korisnika, kojima omogućava bolju interakciju s web stranicama. Analiza zastupljenosti i različite uporabe Web sučelja provedena je u radu [9].

3. TEHNOLOGIJE KORIŠTENE U RADU APLIKACIJSKO PROGRAMSKOG SUČELJA

U ovom poglavlju pružen je kratak osvrt i pojašnjenje nekoliko drugih aplikacija koje su korištene u kreiranju ovog API-ja, bez kojih realizacija samog rada ne bi bila uspješna. Navest će se što sve omogućava svaka od tih tehnologija, u kojem kontekstu su korištene te kako su međusobno povezane.

3.1. phpMyAdmin

phpMyAdmin programski je alat pisan u PHP-u, namijenjen upravljanju rada s MySQL bazom podataka putem Interneta. Podržava brojne operacije vezane uz bazu podataka i tablica koje se nalaze unutar nje, koje se mogu izvesti putem sučelja (pristupačnije i jednostavnije za korisnike koji nisu vješti u pisanju tih operacija) ili putem direktnog izvođenja SQL naredbi napisanih od strane korisnika. Nadalje, moguće je upravljati atributima tablica te birati njihov tip i veličinu, kreirati indekse nad njima, kao i ostaviti dodatne komentare.

U ovom radu, koristi se za kreiranje baze podataka koja sadrži dvije tablice – tablicu s popisom prostorija fakulteta i tablicu sa stavkama inventure. Svaka od njih ima svoj popis atributa, a njihov izgled i svojstva bit će prikazani u idućem poglavlju.

3.2. MAMP

Kako bi se omogućio pristup navedenim bazama podataka, potrebno je uspostaviti lokalni server. Za to služi MAMP. MAMP je program koji omogućava pristup lokalnom PHP i MySQL serveru. Apache komponenta omogućava korisniku pokretanje i održavanje web servera. Pomoću njega je moguće vidjeti i mijenjati vlastite web stranice u pregledniku bez objavljivanja na vanjskom serveru. MySQL komponenta omogućava učitavanje baze podataka te rad s pohranjenim podacima. Svaka od tih komponenti može se instalirati odvojeno, ali MAMP omogućava brži i jednostavniji pristup te grafičko sučelje koje poboljšava preglednost.

U ovom radu, MAMP se koristi za pokretanje lokalnog servera te povezivanje s bazom podataka sadržanom unutar phpMyAdmin-a. Korišteni su već zadani priključi za komponente i lokalni server (Apache port: 8888 i MySQL port: 8889).

3.3. Postman

Nakon uspostavljene veze, potrebno je omogućiti interakciju s podacima. Postman je zajednička platforma namijenjena dizajniranju, razvoju i testiranju API-ja. Sadrži grafičko sučelje koje značajno olakšava testiranje naredbi te pruža uvid u rezultate koji se dobivaju izvođenjem željenih metoda. Umjesto pisanja koda, dovoljno je odabrati željenu HTTP metodu te predati putanju na mjesto adrese. Moguće je odabrati željeni tip ulaznih i izlaznih podataka te uočiti eventualne pogreške u slučaju neispravnosti programskog koda.

U ovom radu, Postman se koristi za dohvaćanje podataka iz tablica u bazi podataka putem GET metode te unošenje novih podataka u iste tablice putem POST metode. Konkretni primjeri i detaljnija objašnjenja dana su u nastavku rada.

Sve tehnologije navedene u ovom poglavlju imaju alternativu te je moguće implementirati ovakvo aplikacijsko programsko sučelje na razne načine. Glavni razlozi odabira ovih tehnologija su pristupačno grafičko sučelje, jednostavna implementacija željenih radnji te olakšano uočavanje pogreške. Bez ovih tehnologija, bilo bi gotovo nemoguće kreirati API, zbog otežane provjere ispravnosti i smislenosti rezultata.

4. REALIZACIJA ZADATKA

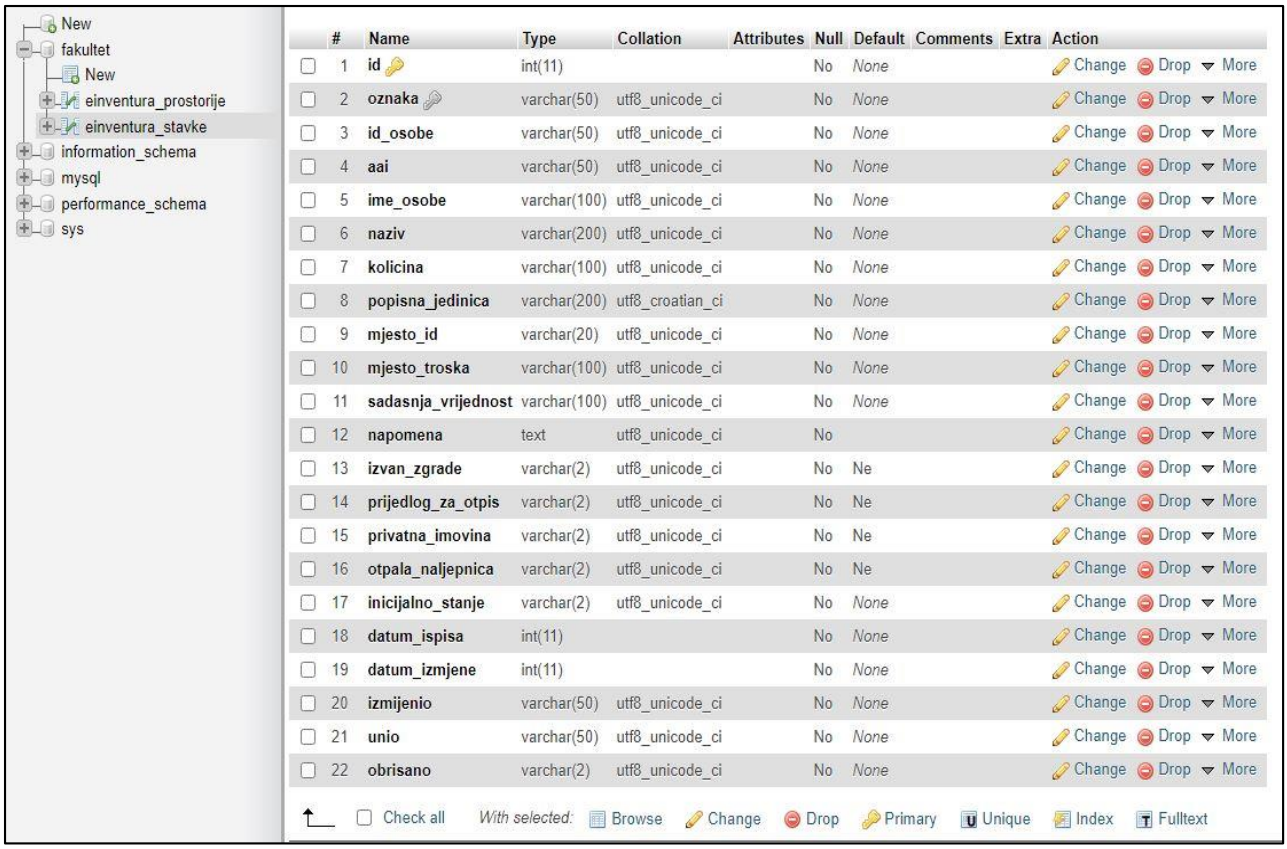
U ovom poglavlju prikazat će se kreiranje aplikacijsko programskog sučelja. Navest će se svi koraci koje je potrebno napraviti prije samog pisanja programskog koda te će se objasniti svaka metoda koju sadrži sučelje i prikazati rezultati njezinog izvođenja.

4.1. Kreiranje i interakcija s bazom podataka

Za sam početak rada, potrebno je kreiranje lokalnog servera. Za taj zadatak koristi se ranije navedena aplikacija MAMP. Komponente koje je potrebno pokrenuti su Apache (omogućava rad sa serverom) i MySQL (omogućava rad s bazom podataka). Za svaku od komponenti koriste se zadani priključci, ali moguće je odabrati proizvoljne vrijednosti, dokle god se nalaze u dozvoljenom rasponu.

Nadalje, korištenjem phpMyAdmin alata, kreirana je baza podataka nazvana „fakultet“. Unutar nje kreirane su dvije tablice – „einventura_prostorije“ i „einventura_stavke“. Atributi tablice, tipovi podataka tih atributa, primarni ključevi i ostala svojstva kreirani su po dobivenim uputama. Obje tablice prikazane su na slikama ispod.

Slika 4.1.: Struktura tablice einvetura_prostorije



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	None			Change Drop More
2	oznaka	varchar(50)	utf8_unicode_ci		No	None			Change Drop More
3	id_osobe	varchar(50)	utf8_unicode_ci		No	None			Change Drop More
4	aai	varchar(50)	utf8_unicode_ci		No	None			Change Drop More
5	ime_osobe	varchar(100)	utf8_unicode_ci		No	None			Change Drop More
6	naziv	varchar(200)	utf8_unicode_ci		No	None			Change Drop More
7	kolicina	varchar(100)	utf8_unicode_ci		No	None			Change Drop More
8	popisna_jedinica	varchar(200)	utf8_croatian_ci		No	None			Change Drop More
9	mjesto_id	varchar(20)	utf8_unicode_ci		No	None			Change Drop More
10	mjesto_troska	varchar(100)	utf8_unicode_ci		No	None			Change Drop More
11	sadasnja_vrijednost	varchar(100)	utf8_unicode_ci		No	None			Change Drop More
12	napomena	text	utf8_unicode_ci		No				Change Drop More
13	izvan_zgrade	varchar(2)	utf8_unicode_ci		No	Ne			Change Drop More
14	prijedlog_za_otpis	varchar(2)	utf8_unicode_ci		No	Ne			Change Drop More
15	privatna_imovina	varchar(2)	utf8_unicode_ci		No	Ne			Change Drop More
16	otpala_naljepnica	varchar(2)	utf8_unicode_ci		No	Ne			Change Drop More
17	inicijalno_stanje	varchar(2)	utf8_unicode_ci		No	None			Change Drop More
18	datum_ispisa	int(11)			No	None			Change Drop More
19	datum_izmjene	int(11)			No	None			Change Drop More
20	izmijenio	varchar(50)	utf8_unicode_ci		No	None			Change Drop More
21	unio	varchar(50)	utf8_unicode_ci		No	None			Change Drop More
22	obrisano	varchar(2)	utf8_unicode_ci		No	None			Change Drop More

Check all With selected: Browse Change Drop Primary Unique Index Fulltext

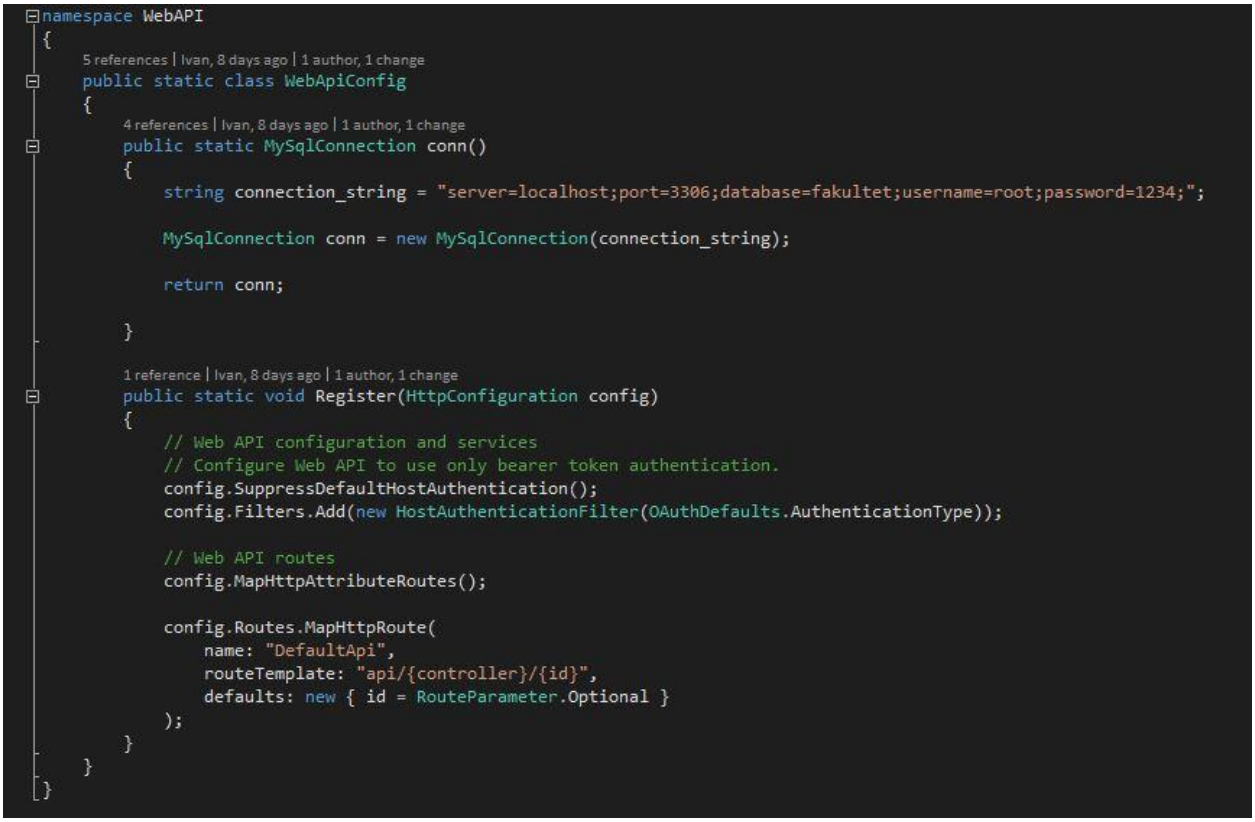
Slika 4.2.: Struktura tablice einvetura_stavke

Tablica prikazana slikom 4.1. popunjena je stvarnim podacima o prostorijama na fakultetu, dok je tablica na slici 4.2. popunjena proizvoljnim vrijednostima.

Za kreiranje samog projekta korišten je predloženi Web API template iz .NET Frameworka. Pri kreiranju projekta dodan je MySQL C# connector – poveznica koja omogućuje da .NET aplikacije komuniciraju s MySQL serverima. Implementira se tako što se na početku programa dodaje „using MySQL.Data.MySqlClient;“.

Unutar projekta kreirana je klasa „Stavka“, čiji se atributi poklapaju s onima iz tablice u bazi podataka. Također, implementirana su dva konstruktora, jedan bez parametara i jedan koji ih posjeduje, kako bi se omogućilo spremanje podataka iz baze u objekte klase. Ovisno o korištenoj metodi, radit će se s jednim objektom ili s listom objekata.

Zadnje što je potrebno napraviti prije implementacije metoda je povezati projekt s bazom podataka. Veza se kreira u dokumentu „WebApiConfig.cs“ uz pomoć priključaka navedenih u MAMP aplikaciji. Također, i ovdje je potrebno dodati „using MySQL.Data.MySqlClient;“. Na slici 4.3. prikazan je programski kod iz navedenog dokumenta.



Slika 4.3.: Ostvarivanje veze s bazom podataka unutar WebApiConfig klase

Unutar metode Register, koja je sastavni dio navedene klase, ne vrše se nikakve promjene, već se kreira nova metoda nazvana conn(), koja vraća vezu na bazu podataka. Unutar metode kreirana je string varijabla koja sadrži potrebne informacije za kreiranje veze. Sadrži podatke o pokrenutom serveru, MySQL priključku putem kojeg se povezuje na bazu, naziv baze podataka na koju se spaja te informacije o korisničkom imenu i šifri korisnika. Informacije o priključku, korisničkom imenu i šifri mogu se pronaći unutar MAMP aplikacije. Ta varijabla predaje se kao parametar u kreiranju nove veze, koja se kasnije šalje kao povratna vrijednost metode. Moguće je kreirati novu vezu direktnim upisivanjem teksta u argument konstruktora, ali je zbog preglednosti korištena dodatna varijabla. Ova metoda koristit će se u svakoj HTTP funkciji obrađenoj u glavnom dijelu programa, pri kreiranju veze s bazom podataka.

4.2. Get () metoda

Get() metoda zadužena je za vraćanje liste svih stavki inventure iz tablice u bazi podataka. Način na koji se poziva ova metoda je odabirom GET HTTP metode te završavanjem URL putanje s /api/values. Implementacija metoda prikazana je na slici 4.4.

```
// GET api/values
0 references | Ivan, 7 days ago | 1 author, 2 changes
public List<stavka> Get()
{
    MySqlConnection connection = WebApiConfig.conn();

    MySqlCommand query = connection.CreateCommand();



    query.CommandText = "SELECT * FROM einventura_stavke";

    var stavka = new List<stavka>();

    try
    {
        connection.Open();
    }
    catch (MySql.Data.MySqlClient.MySqlException exception)
    {
        throw exception;
    }



    MySqlDataReader fetch_query = query.ExecuteReader();


    while (fetch_query.Read())
    {
        stavka.Add(new stavka(int.Parse( fetch_query["id"].ToString()), fetch_query["oznaka"].ToString(), fetch_query["id_osobe"].ToString(), fetch_query["aai"].ToString(),
                              fetch_query["ime_osobe"].ToString(), fetch_query["naziv"].ToString(), fetch_query["kolicina"].ToString(), fetch_query["popisna_jedinica"].ToString(),
                              fetch_query["mjesto_id"].ToString(), fetch_query["mjesto_troska"].ToString(), fetch_query["sadasnja_vrijednost"].ToString(), fetch_query["napomena"].ToString(),
                              fetch_query["izvan_zgrade"].ToString(), fetch_query["prijedlog_za_otpis"].ToString(), fetch_query["privatna_imovina"].ToString(),
                              fetch_query["otpala_naljepnica"].ToString(), fetch_query["inicijalno_stanje"].ToString(), int.Parse(fetch_query["datum_ispisa"].ToString()),
                              int.Parse(fetch_query["datum_izmjene"].ToString()), fetch_query["izmijenio"].ToString(), fetch_query["unio"].ToString(),fetch_query["obrisano"].ToString()));
    }

    return stavka;

}
```

Slika 4.4.: Implementacija Get() metode

Putem ranije opisane funkcije conn(), kreira se veza s bazom podataka. Nakon toga kreira se upit koji će sadržavati SQL naredbu. U ovom slučaju koristi se naredba SELECT *, pomoću koje se odabiru svi podaci iz tablice einventura_stavke. Kreira se lista čiji su elementi objekti klase Stavka te se otvara veza prema bazi podataka putem operacije connection.Open(). Nakon što je otvorena veza, započinje čitanje podataka iz tablice. Do toga dolazi izvršavanjem upita koji sadrži SQL naredbu za odabir svih elemenata. Unutar while petlje, u listu stavki dodaju se novi objekti iz tablice dok se ne dođe do njezinog kraja. Za stvaranje novih objekata koristi se konstruktor s parametrima, a svaki parametar izjednačava se s odgovarajućim atributom iz tablice (zbog toga je bitno da su tablica stavki i klasa stavki napravljena u projektu identične). Nakon što spremi sve stavke iz tablice u listu, metoda vraća listu kao povratnu vrijednost. Na slici 4.5. prikazana je tablica u bazi podataka koja sadrži proizvoljne vrijednosti stavki inventure, dok slika 4.6. prikazuje rezultat izvršavanja ove metode unutar Postman platforme.

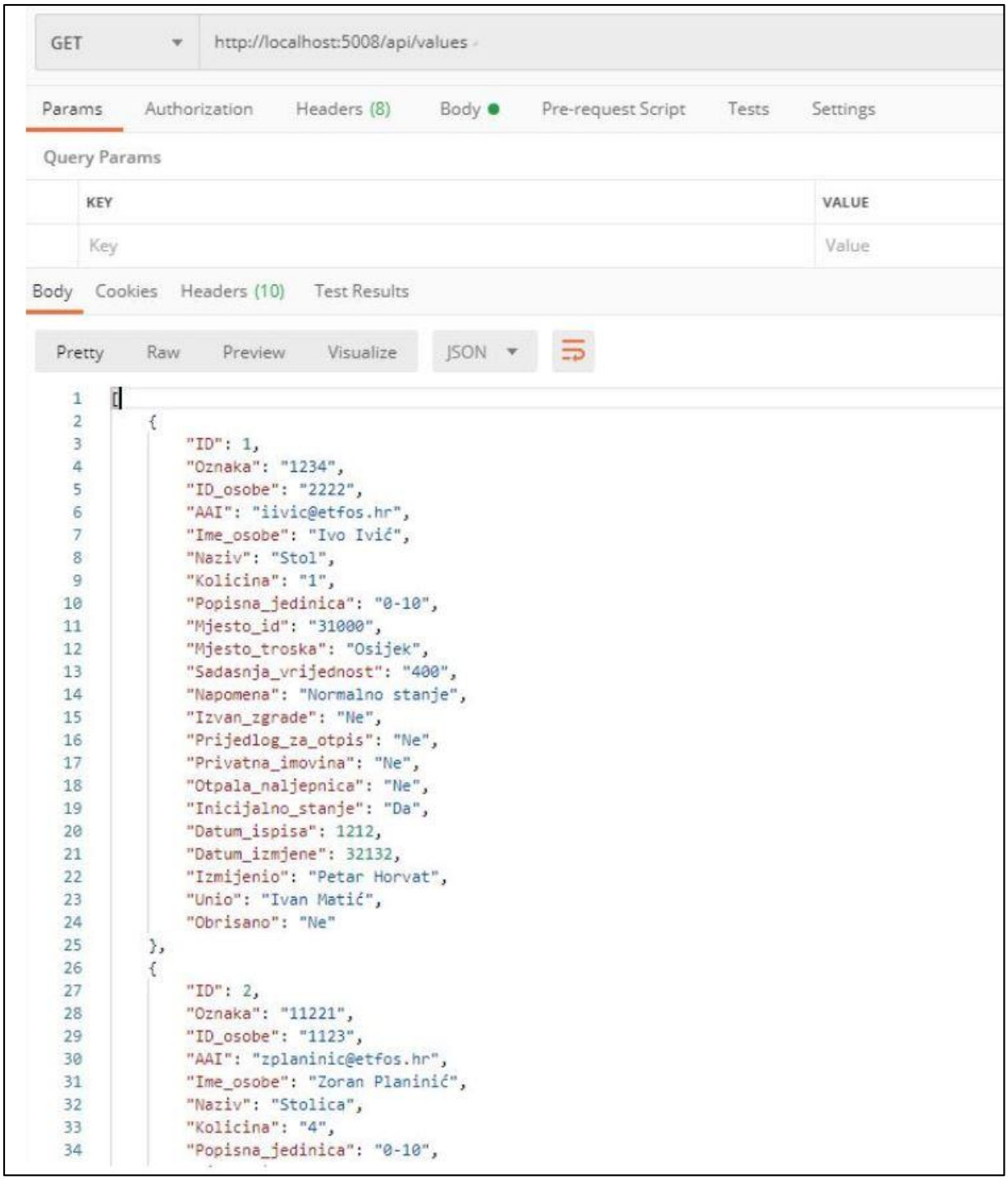
+ Options

	id	oznaka	id_osobe	aai	ime_osobe	naziv	kolicina	popisna_jedinica
Edit Copy Delete	1	1234	2222	iivic@etfos.hr	Ivo Ivić	Stol	1	0-10
Edit Copy Delete	2	11221	1123	zplaninic@etfos.hr	Zoran Planinić	Stolica	4	0-10
Edit Copy Delete	3	1111111	2223	jtudor@etfos.hr	Juraj Tudor	Računalo	3	3-51
Edit Copy Delete	4	199	99	jperic@etfos.hr	Juraj Perić	Svjetiljka	1	2-50
Edit Copy Delete	5	12334	22212	imarkovic@etfos.hr	Ivan Markovic	Projektor	1	2-50
Edit Copy Delete	6	1214	82212	imak@etfos.hr	Ivana Mak	Pisač	1	1-52
Edit Copy Delete	7	6214	12212	imaric@etfos.hr	Iva Maric	Ormar	2	1-51
Edit Copy Delete	8	198214	12212	ckovac@etfos.hr	Cvjetko Kovač	Ormar	2	1-51
Edit Copy Delete	9	15214	18212	mtopic@etfos.hr	Mihovil Topić	Stolica	1	2-50
Edit Copy Delete	10	45214	18212	nmatkovic@etfos.hr	Nives Matković	Televizor	1	0-34
Edit Copy Delete	11	6545214	18212	rmarjanovic@etfos.hr	Ruža Marjanović	Pisač	1	3-50
Edit Copy Delete	12	0545214	18212	ialilovic@etfos.hr	Ivo Alilovic	Projektor	2	0-10
Edit Copy Delete	13	44345214	38212	ppavicic@etfos.hr	Petar Pavičić	Stol	2	1-54

Check all *With selected:* Edit Copy Delete Export

Slika 4.5.: Tablica einventura_stavke

Tablica koja prikazuje stavke inventure popunjena je proizvoljnim podacima, a koristit će se u nastavku radu za provjeru točnosti rezultata metoda. Na slici 4.5. nisu prikazani svi atributi tablice, ali jesu oni po kojima će se vršiti određeno filtriranje stavki – id stavke i popisna jedinica. Tablica je popunjena unutar phpMyAdmin alata putem sučelja radi jednostavnosti, ali istu operaciju moguće je izvršiti putem pisane SQL naredbe ili korištenjem Post () metode, koja će biti obrađena u nastavku rada.



Slika 4.6.: Izvršavanje Get() metode unutar Postman platforme

Unutar dijela s adresom, vidljivo je korištenje GET metode te da URL putanja završava na potreban način. Rezultat je vraćen kao polje JSON objekata te je moguće iščitati vrijednosti bilo kojeg parametra za svaki objekt. U nastavku će se pokazati kako se ova funkcija, uz određene izmjene, može koristiti i za dohvaćanje samo onih stavki koje zadovoljavaju određene uvjete pretraživanja.

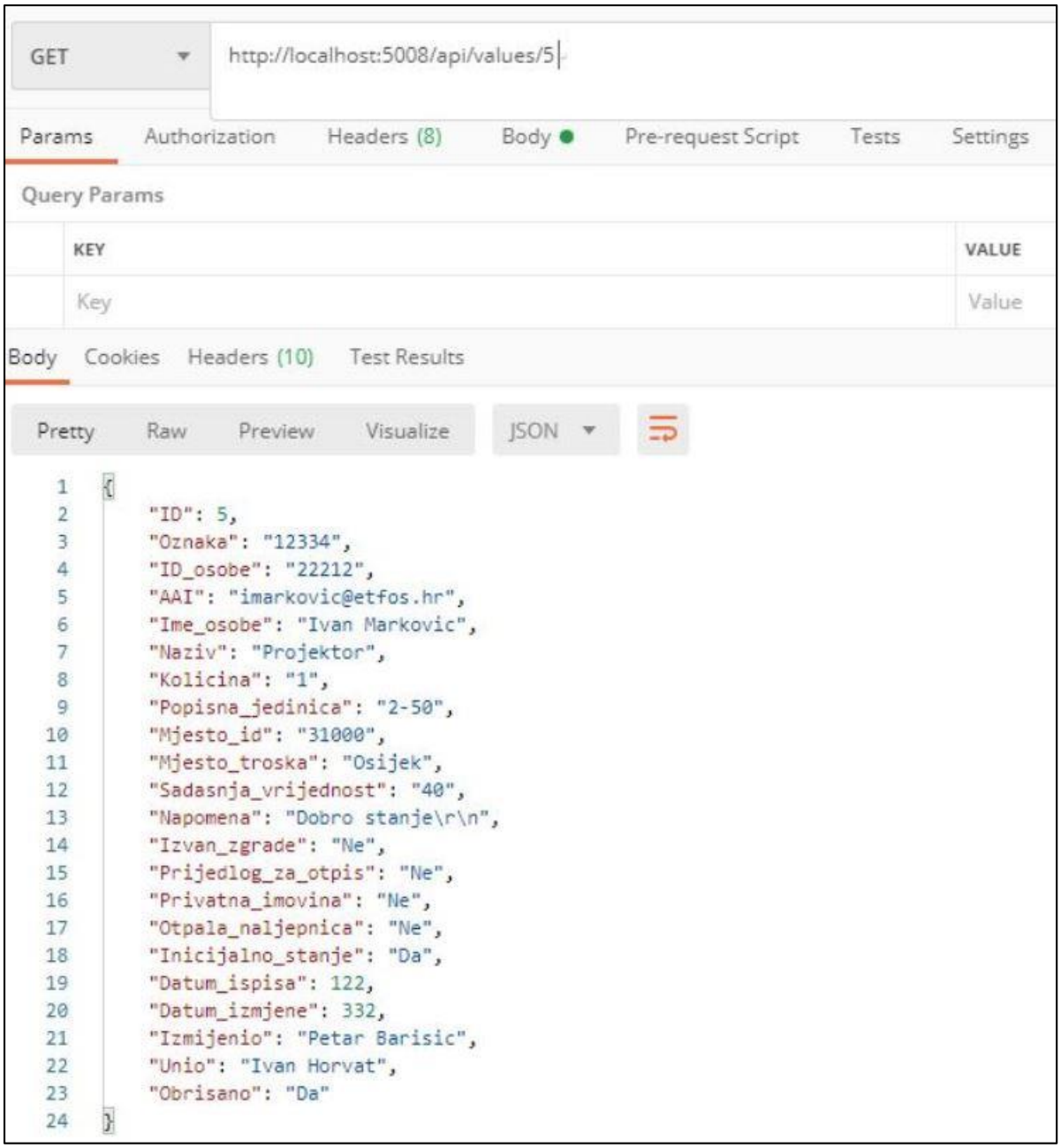
4.3. Get (int id) metoda

Get(int id) metoda zadužena je za vraćanje samo jednog objekta iz tablice stavki inventure, čiji primarni ključ odgovara predanom parametru metode. Način na koji se poziva ova metoda je odabirom GET HTTP metode te završavanjem URL putanje s /api/values/id. Implementacija metoda prikazana je na slici 4.7.

```
// GET api/values/5
[Route("{id:int}")]
0 references | Ivan, 7 days ago | 1 author, 2 changes
public stavka Get(int id)
{
    MySqlConnection connection = WebApiConfig.conn();

    MySqlCommand query = connection.CreateCommand();

    query.CommandText = "SELECT * FROM einventura_stavke, einventura_prostorije " +
                        "WHERE einventura_stavke.popisna_jedinica = einventura_prostorije.popisna_jedinica AND einventura_stavke.id = @id";

    query.Parameters.AddWithValue("@id", id);

    var stavka = new List<stavka>();

    try
    {
        connection.Open();
    }
    catch (MySql.Data.MySqlClient.MySqlException exception)
    {
        throw exception;
    }



    var odabrana_stavka = new stavka();

    MySqlDataReader fetch_query = query.ExecuteReader();


    while (fetch_query.Read())
    {
        odabrana_stavka = new stavka(int.Parse(fetch_query["id"].ToString()), fetch_query["oznaka"].ToString(), fetch_query["id_osobe"].ToString(), fetch_query["aai"].ToString(),
                        fetch_query["ime_osobe"].ToString(), fetch_query["naziv"].ToString(), fetch_query["kolicina"].ToString(), fetch_query["popisna_jedinica"].ToString(),
                        fetch_query["mjesto_id"].ToString(), fetch_query["mjesto_troska"].ToString(), fetch_query["sadasnja_vrijednost"].ToString(), fetch_query["napomena"].ToString(),
                        fetch_query["izvan_zgrade"].ToString(), fetch_query["prijedlog_za_otpis"].ToString(), fetch_query["privatna_imovina"].ToString(),
                        fetch_query["otpala_naljepnica"].ToString(), fetch_query["inicijalno_stanje"].ToString(), int.Parse(fetch_query["datum_ispisa"].ToString()),
                        int.Parse(fetch_query["datum_izmjene"].ToString()), fetch_query["izmijenio"].ToString(), fetch_query["unio"].ToString(), fetch_query["obrisano"].ToString());
    }

    return odabrana_stavka;

}
```

Slika 4.7.: Implementacija Get(int id) metode

Uspostavljanje veze i kreiranje upita vrše se na identičan način kao u prethodnoj metodi. Kako bi se razlikovala od metode koja će biti obrađena u nastavku rada, iznad same metode dodaje se specifikacija putanje, u kojoj se navodi da je krajnji element id (poklapa se s nazivom argumenta metode) i da je njegov tip podatka integer. Nadalje, rade se određene izmjene u SQL naredbi. I dalje se koristi naredba SELECT *, ali na kraju se dodaje uvjet u kojem se zahtjeva da se vrati samo stavka čiji atribut ID odgovara argumentu metode. Kako bi se navedeni argument mogao unijeti u SQL naredbu, koristi se metoda .AddWithValue(), u kojoj se definira da dio unutar uvjeta označen s @id poprima vrijednost argumenta. Nadalje, kreira se objekt klase Stavka te se otvara veza prema bazi podataka putem operacije connection.Open(). Nakon što je otvorena veza, započinje se čitanje podataka iz tablice na ranije opisan način. Unutar while petlje, kreirani objekt klase izjednačava se s odgovarajućom stavkom iz tablice te se vraća kao povratna vrijednost funkcije. Na slici 4.8. prikazan je rezultat izvršavanja ove metode unutar Postman platforme.



Slika 4.8.: Izvršavanje Get(int id) metode unutar Postman platforme

Unutar dijela s adresom, uočava se korištenje GET metode te da URL putanja završava na potreban način. Za testiranje je kao krajnji dio putanje predan id čija je vrijednost 5, a rezultat je vraćen kao JSON objekt čija je vrijednost ID parametra također 5. Ako se dobiveni rezultat usporedi s vrijednostima u tablici prikazanoj na slici 4.5., uočava se poklapanje rezultata tj. da je dohvaćena peta stavka iz tablice. Ova metoda korisna je pri radu s predmetima koji imaju jedinstveni identifikator, dok će se u nastavku obraditi metoda za dohvaćanje više predmeta koji ispunjavaju određeni uvjet.

4.4. Get (string prostorija) metoda

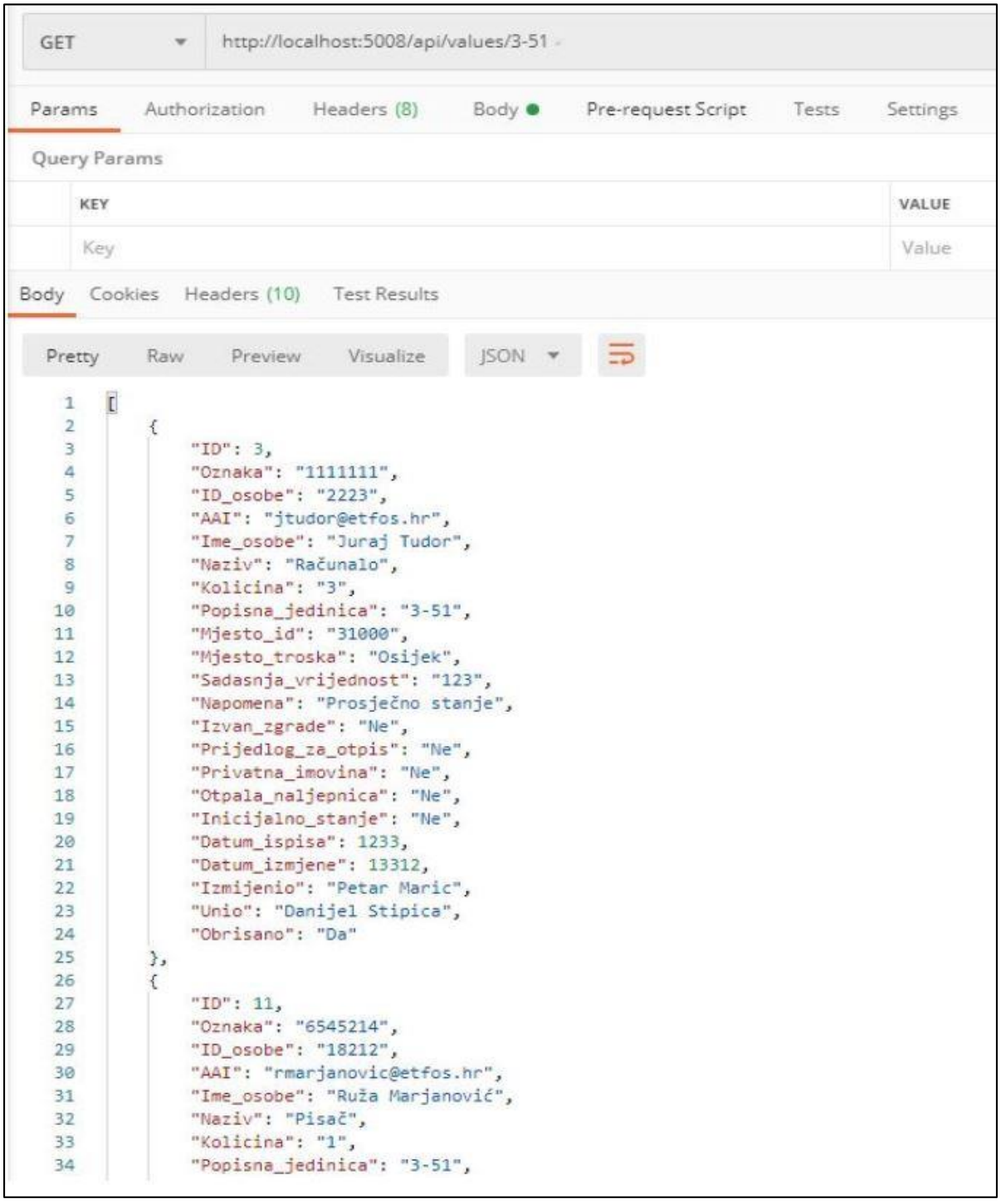
Get(string prostorija) metoda zadužena je za vraćanje liste svih stavki inventure iz tablice čija se popisna jedinica poklapa s traženom prostorijom iz argumenta metode. Način na koji se poziva ova metoda je odabirom GET HTTP metode te završavanjem URL putanje s /api/values/prostorija. Implementacija metoda prikazana je na slici 4.9.

```
// GET api/values/0-10
[Route("{prostorija}")]
0 references | Ivan, 7 days ago | 1 author, 1 change
public List<stavka> Get(string prostorija)
{
    MySqlConnection connection = WebApiConfig.conn();

    MySqlCommand query = connection.CreateCommand();

    query.CommandText = "SELECT * FROM einventura_stavke WHERE popisna_jedinica = @prostorija";

    query.Parameters.AddWithValue("@prostorija", prostorija);

    var stavka = new List<stavka>();

    try
    {
        connection.Open();
    }
    catch (MySql.Data.MySqlClient.MySqlException exception)
    {
        throw exception;
    }



    MySqlDataReader fetch_query = query.ExecuteReader();


    while (fetch_query.Read())
    {
        stavka.Add(new stavka(int.Parse(fetch_query["id"].ToString()), fetch_query["oznaka"].ToString(), fetch_query["id_osobe"].ToString(), fetch_query["aai"].ToString(),
                              fetch_query["ime_osobe"].ToString(), fetch_query["naziv"].ToString(), fetch_query["kolicina"].ToString(), fetch_query["popisna_jedinica"].ToString(),
                              fetch_query["mjesto_id"].ToString(), fetch_query["mjesto_troska"].ToString(), fetch_query["sadasnja_vrijednost"].ToString(), fetch_query["napomena"].ToString(),
                              fetch_query["izvan_zgrade"].ToString(), fetch_query["prijedlog_za_otpis"].ToString(), fetch_query["privatna_imovina"].ToString(),
                              fetch_query["otpala_naljepnica"].ToString(), fetch_query["inicijalno_stanje"].ToString(), int.Parse(fetch_query["datum_ispisa"].ToString()),
                              int.Parse(fetch_query["datum_izmjene"].ToString()), fetch_query["izmijenio"].ToString(), fetch_query["unio"].ToString(), fetch_query["obrisano"].ToString()));
    }

    return stavka;

}
```

Slika 4.9.: Implementacija Get(string prostorija) metode

Uspostavljanje veze i kreiranje upita vrši se na identičan način kao u prethodnim metodama. Kako bi se razlikovala od metode koja dohvaća samo onu stavku inventure s odgovarajućim primarnim ključem prostoriji te se na kraju izvršavanja SQL naredbe ta lista vraća kao povratna vrijednost funkcije. Na slici 4.10. prikazan je rezultat izvršavanja ove metode unutar Postman platforme., iznad same metode dodaje se specifikacija putanje, u kojoj se navodi da je krajnji element prostorija(poklapa se s nazivom argumenta metode). U ovom slučaju ne navodi se da je tip podatka string jer je to automatski zadan tip. Ponovno se rade određene izmjene u SQL naredbi. I dalje se koristi naredba SELECT *, ali na kraju se dodaje uvjet u kojem se zahtjeva da se vrate samo stavke čiji atribut popisna jedinica odgovara predanoj prostoriji. Kako bi se navedeni argument mogao unijeti u SQL naredbu, ponovno se koristi metoda .AddWithValue(), u kojoj se definira da dio unutar uvjeta označen s @prostorija poprima vrijednost argumenta. Kreira se lista objekata klase Stavka te se otvara veza prema bazi podataka putem operacije connection.Open(). Nakon što je

otvorena veza, započinje se s čitanjem podataka iz tablice na ranije opisan način. Unutar while petlje, navedena lista objekata popunjava se elementima tablice koji se nalaze u odgovarajućoj



Slika 4.10.: Izvršavanje Get(string prostorija) metode unutar Postman platforme

Unutar dijela s adresom, uočava se korištenje GET metode te da URL putanja završava na potreban način. Za testiranje je kao krajnji dio putanje predana prostorija 3-51, a rezultati koji su vraćeni unutar polja JSON objekata imaju tu prostoriju pod parametrom Popisna_jedinica. Ako se

dobiveni rezultati usporede s vrijednostima u tablici prikazanoj na slici 4.5., uočava se da je dobivena lista ispravna tj. da su dohvaćene sve stavke koje se nalaze u toj prostoriji. Ova metoda korisna je za filtriranje elemenata tablice po željenim uvjetima.

4.5. Post (stavka nova_stavka) metoda

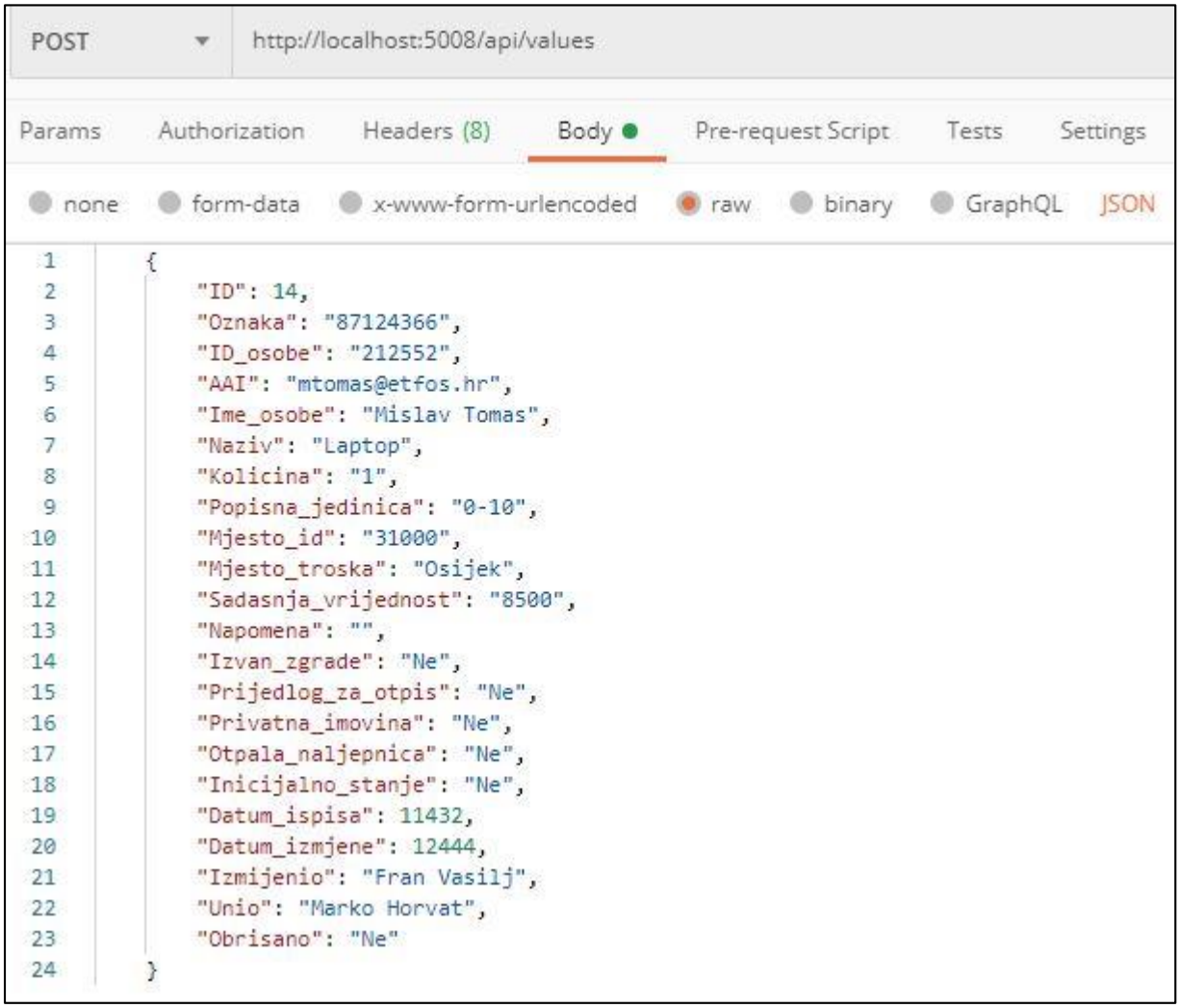
Na kraju, prikazat će se implementacija Post(stavka nova_stavka) metode. Za razliku od dosadašnjih metoda, ova nema povratnu vrijednost, već je zadužena za umetanje novih podataka u tablicu. Također, moraju se uvesti određene promjene pri navođenju argumenta funkcije te u radu s Postman platformom. Način na koji se poziva ova metoda je odabirom POST HTTP metode te završavanjem URL putanje s /api/values. Implementacija metoda prikazana je na slici 4.11.

```
// POST api/values
[HttpPost]
0 references | Ivan, 7 days ago | 1 author, 2 changes
public void Post([FromBody] stavka stavka)
{
    MySqlConnection connection = WebApiConfig.conn();
    MySqlCommand query = connection.CreateCommand();

    query.CommandText = "INSERT INTO einventura_stavke (id, oznaka, id_osobe, aai, ime_osobe, naziv, kolicina, popisna_jedinica, mjesto_id, mjesto_troska, "+
                        "sadasnja_vrijednost, napomena, izvan_zgrade, prijedlog_za_otpis, privatna_imovina, otpala_naljepnica, inicijalno_stanje, " +
                        "datum_ispisa, datum_izmjene, izmijenio, unio, obrisano) "+
                       "VALUES (@id, @oznaka, @id_osobe, @aai, @ime_osobe, @naziv, @kolicina, @popisna_jedinica, @mjesto_id, "+
                       "@mjesto_troska, @sadasnja_vrijednost, @napomena, @izvan_zgrade, @prijedlog_za_otpis, @privatna_imovina, @otpala_naljepnica, "+
                       "@inicijalno_stanje, @datum_ispisa, @datum_izmjene, @izmijenio, @unio, @obrisano)";

    query.Parameters.AddWithValue("@id", stavka.ID);
    query.Parameters.AddWithValue("@oznaka", stavka.Oznaka);
    query.Parameters.AddWithValue("@id_osobe", stavka.ID_osobe);
    query.Parameters.AddWithValue("@aai", stavka.AAI);
    query.Parameters.AddWithValue("@ime_osobe", stavka.Ime_osobe);
    query.Parameters.AddWithValue("@naziv", stavka.Naziv);
    query.Parameters.AddWithValue("@kolicina", stavka.Kolicina);
    query.Parameters.AddWithValue("@popisna_jedinica", stavka.Popisna_jedinica);
    query.Parameters.AddWithValue("@mjesto_id", stavka.Mjesto_id);
    query.Parameters.AddWithValue("@mjesto_troska", stavka.Mjesto_troska);
    query.Parameters.AddWithValue("@sadasnja_vrijednost", stavka.Sadasnja_vrijednost);
    query.Parameters.AddWithValue("@napomena", stavka.Napomena);
    query.Parameters.AddWithValue("@izvan_zgrade", stavka.Izvan_zgrade);
    query.Parameters.AddWithValue("@prijedlog_za_otpis", stavka.Prijedlog_za_otpis);
    query.Parameters.AddWithValue("@privatna_imovina", stavka.Privatna_imovina);
    query.Parameters.AddWithValue("@otpala_naljepnica", stavka.Otpala_naljepnica);
    query.Parameters.AddWithValue("@inicijalno_stanje", stavka.Inicijalno_stanje);
    query.Parameters.AddWithValue("@datum_ispisa", stavka.Datum_ispisa);
    query.Parameters.AddWithValue("@datum_izmjene", stavka.Datum_izmjene);
    query.Parameters.AddWithValue("@izmijenio", stavka.Izmijenio);
    query.Parameters.AddWithValue("@unio", stavka.Unio);
    query.Parameters.AddWithValue("@obrisano", stavka.Obrisano);

    try{
        connection.Open();
    }
    catch (MySql.Data.MySqlClient.MySqlException exception){
        throw exception;
    }

    try{
        MySqlDataReader fetch_query = query.ExecuteReader();
    }
    catch (MySql.Data.MySqlClient.MySqlException exception){
        throw exception;
    }
}
```

Slika 4.11.: Implementacija Post(stavka nova_stavka) metode

Prvu izmjenu moguće je uočiti odmah pri upisivanju parametra metode. Navođenjem [FromBody] atributa ispred argumenta metode, daje se uputa Web API-ju da traženu stavku treba iščitati iz

body dijela primljenog zahtjeva (eng. *request body*). U ovom slučaju, to se odnosi na Body karticu unutar Postman platforme, koja će se detaljnije opisati pri analizi rezultata. Uspostavljanje veze i kreiranje upita vrši se na identičan način kao u dosadašnjim metodama, bez obzira što se koristi druga HTTP metoda. U ovoj metodi koristi se drugačija SQL naredba. Umjesto dosadašnje SELECT *, upisuje se INSERT INTO ime_tablice (stupci tablice) VALUES (željene vrijednosti), te se navodi tablica einvetura_stavke kao mjesto na koje unosimo podatke. U nastavku naredbe navode se svi parametri iz stavke koja je predana kao argument. Ponovno se koristi metoda .AddWithValue() kako bi se svaka od @ oznaka iz naredbe zamijenila s atributom iz stavke inventure. Zbog preglednosti, ovaj dio programskog koda napisan je jedan ispod drugog. Otvara se veza prema bazi podataka putem operacije connection.Open() te se izvršava upit koji sadrži navedenu SQL naredbu. Na slici 4.12. prikazano je obavljanje Post() metode unutar Postman platforme, dok slika 4.13. prikazuje promjene u tablici baze podataka.



Slika 4.12.: Izvršavanje Post() metode unutar Postman platforme

Unutar dijela s adresom, uočava se korištenje POST metode te da URL putanja završava na potreban način. S obzirom da je unutar argumenata funkcije naveden atribut [FromBody], novi podatak u Postmanu piše se unutar Body kartice. Odabire se da će podatak biti napisan kao JSON objekt te se unose parametri u parovima ključ/vrijednost (eng. *key/value*). Pri unošenju, mora se paziti da vrijednost parametra ID već ne postoji u tablici, jer se radi o primarnom ključu, te se operacija neće uspješno izvesti. Isto vrijedi i za parametar Oznaka, koji je jedinstveni ključ. Kada se unesu svi podaci, klikom na naredbu Send šalju se u bazu podataka.

+ Options

	id	oznaka	id_osobe	aai	ime_osobe	naziv	kolicina	popisna_jedinica
Edit Copy Delete	1	1234	2222	iivic@etfos.hr	Ivo Ivić	Stol	1	0-10
Edit Copy Delete	2	11221	1123	zplaninic@etfos.hr	Zoran Planinić	Stolica	4	0-10
Edit Copy Delete	3	1111111	2223	jtudor@etfos.hr	Juraj Tudor	Računalo	3	3-51
Edit Copy Delete	4	199	99	jperic@etfos.hr	Juraj Perić	Svjetiljka	1	2-50
Edit Copy Delete	5	12334	22212	imarkovic@etfos.hr	Ivan Markovic	Projektor	1	2-50
Edit Copy Delete	6	1214	82212	imak@etfos.hr	Ivana Mak	Pisač	1	1-52
Edit Copy Delete	7	6214	12212	imaric@etfos.hr	Iva Maric	Ormar	2	1-51
Edit Copy Delete	8	198214	12212	ckovac@etfos.hr	Cvjetko Kovač	Ormar	2	1-51
Edit Copy Delete	9	15214	18212	mtopic@etfos.hr	Mihovil Topić	Stolica	1	2-50
Edit Copy Delete	10	45214	18212	nmatkovic@etfos.hr	Nives Matković	Televizor	1	0-34
Edit Copy Delete	11	6545214	18212	rmarjanovic@etfos.hr	Ruža Marjanović	Pisač	1	3-51
Edit Copy Delete	12	0545214	18212	ialilovic@etfos.hr	Ivo Alilovic	Projektor	2	0-10
Edit Copy Delete	13	44345214	38212	ppavicic@etfos.hr	Petar Pavičić	Stol	2	1-54
Edit Copy Delete	14	87124366	212552	mtomas@etfos.hr	Mislav Tomas	Laptop	1	0-10

Slika 4.13.: Tablica einventura_stavke nakon izvođenja Post() metode

U tablici prikazanoj na slici iznad, uočava se kako je naredba sa slike 4.12. uspješno izvedena. Dodan je novi element u tablicu (prikazanu slikom 4.5.) i svi parametri navedeni u JSON objektu uneseni su u odgovarajući stupac tablice. Ovu metodu moguće je doraditi tako da omogući i unošenje više novih stavki odjednom (predajom liste ili polja objekata kao argument funkcije), ali u ovom radu takva metoda nije implementirana.

Obrađene metode i rezultati njihovog izvođenja daju uvid u funkcioniranje aplikacijsko programskog sučelja. Najčešći uzrok neispravnosti u radu su greške pri pisanju SQL naredbe (uglavnom nastaju zbog količine teksta i otežane preglednosti) pa se savjetuje provjera svih parametara te pisanje programskog koda u lakše čitljivom obliku, kako bi se te pogreške brže uočile. Također, preporuča se korištenje Try-Catch blokova pri uspostavljanju veze s bazom podataka te radom s njezinim tablicama.

5. ZAKLJUČAK

U ovom završnom radu prikazana je realizacija aplikacijsko programskog sučelja namijenjenog radu s bazama podataka. Prikazani su svi potrebni koraci u kreaciji funkcionalnog sučelja – od uspostavljanja lokalnog servera i kreacije baze podataka do izvođenja implementiranih funkcija putem platforme za slanje zahtjeva. Navedene su i ukratko objašnjene sve aplikacije koje se koriste u realizaciji projekta, a nisu dio sučelja. Nadalje, opisane su sve metode koje se nalaze unutar sučelja te način na koji one rade. Također, komentirani su rezultati njihovog izvođenja.

Aplikacijsko programsko sučelje napravljeno u ovom završnom radu može se primijeniti, uz određene izmjene, u realizaciji različitih projekata koji rade sa strukturiranim bazama podataka. Prilagodba na tablice s drugačijim atributima ili na drugačije parametre po kojima se vrši pretraživanje, trebala bi biti dovoljna za uporabu u drugim projektima.

Ovaj rad moguće je dodatno proširiti implementacijom HTTP metoda poput PUT i DELETE te proširenjem POST metode ranije navedenim primjerom, u kojem se omogućava dodavanje više elemenata istovremeno. Nadalje, ovo sučelje moguće je realizirati u drugim programskim jezicima, ovisno koji jezik preferira programer koji ga izvodi, bez da se gubi funkcionalnost. Slično tome, mogu se koristiti druge aplikacije za kreiranje servera i uspostavljanje veze te za slanje zahtjeva prema bazi podataka. Najveća prednost ovog projekta je što odvaja sve metode zadužene za rad s webom na jedno mjesto te omogućava njihovo daljnje korištenje bez potrebe da drugi korisnici znaju na koji su način implementirane.

LITERATURA

[1] M. Biehl, API Architecture: The Big Picture for Building APIs, CreateSpace, svibanj 2015.

[2] J. Tulach, Practical API Design: Confessions of a Java Framework Architect, Apress, srpanj 2008.

[3] B. Jin, S. Sahni, A Shevat, Designing Web APIs: Building APIs That Developers Love, O'Reilly Media, Sebastopol, CA, rujan 2018.

[4] J. Kurtz, B. Wortman, ASP.NET Web API 2: Building a REST Service from Start to Finish, Apress, srpanj 2014.

[5] P. Ferguson, R. Eckstein, Java™ in a Nutshell, Fourth Edition, O'Reilly & Associates, Inc., Sebastopol, CA, ožujak 2002.

[6] F. Doglio, Pro REST API Development with Node.js, Apress, New York, svibanj 2015.

[7] J. Elman, M. Lavin, Lightweight Django, O'Reilly Media, studeni 2014.

[8] M. Reddy, API Design for C++, Elsevier, veljača 2011.

[9] J. Wagner: The Increasing Importance of APIs in Web Development https://code.tutsplus.com/articles/the-increasing-importance-of-apis-in-web-development--net-22368 (pristupljeno 7.7.2020.)

SAŽETAK

U završnom radu kreiran je Web API namijenjen provođenju inventure stavki na fakultetu. Sučelje je implementirano u C# programskom jeziku i .NET Frameworku, ali su navedeni i drugi programski jezici te literatura koja ih detaljnije objašnjava. Ovo sučelje kreirano je za rad s bazama podataka. U teoretskom uvodu opisane su dodatne aplikacije korištene za olakšavanje rada sučelja te koje sve prednosti one donose. Praktični dio bavi se implementacijom samih metoda te objašnjavanjem njihovog programskog koda. Na kraju svake metode prikazuju se i komentiraju njezini rezultati.

Ključne riječi: API, aplikacijsko programsko sučelje, baze podataka, HTTP metode

ABSTRACT

CREATION OF APP APPLICATION PROGRAMMING INTERFACE FOR INVENTORY CHECKING

In this bachelor thesis, a Web API meant for inventory checking items in faculty offices was developed. Interface was implemented in C# programming language and .NET Framework, but there are other programming languages mentioned in the thesis, together with literature providing further explanation. This interface was created for working with databases. In theoretical part, other applications used for making the usage of interface easier were described. Also, all the benefits that they bring are mentioned. Practical part of this thesis shows implementation of HTTP methods and explains coding behind them. At the end of each method, their results are shown and discussed.

Keywords: API, application programming interface, database, HTTP methods