Izrada BaaS (Backend as a Service) sustava za web aplikacije

Dudjak, Mario

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

Sveučilišni studij

IZRADA BACKEND AS A SERVICE SUSTAVA ZA WEB APLIKACIJE

Diplomski rad

Mario Dudjak

Osijek, 2018.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 05.09.2018.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Mario Dudjak
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 878 R, 20.09.2017.
OIB studenta:	03714569831
Mentor:	Prof.dr.sc. Goran Martinović
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Alfonzo Baumgartner
Član Povjerenstva:	Dr.sc. Bruno Zorić
Naslov diplomskog rada:	Izrada BaaS (Backend as a Service) sustava za web aplikacije
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U diplomskom radu potrebno je opisati postupak izrade i mogućnosti BaaS (Backend as a Service) sustava. U praktičnom dijelu rada treba programski izraditi određeni broj web API-ja koji pokrivaju najčešće funkcionalnosti današnjih web sustava. Njih treba integrirati u SDK za neku od okolina za izradu web aplikacije, te podržati odgovarajućim IaaS i PaaS komponentama. Prikladnost korištenja razvijenog BaaS sustava treba pokazati kroz izradu web aplikacije koja koristi funkcionalnosti omogućene sustavom. Razvijenu aplikaciju treba prikladno testirati i analizirati.
Prijedlog ocjene pismenog dijela ispita (diplomskog 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: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	05.09.2018.

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, 13.09.2018.

Ime i prezime studenta:	Mario Dudjak
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 878 R, 20.09.2017.
Ephorus podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Izrada BaaS (Backend as a Service) sustava za web aplikacije**

izrađen pod vodstvom mentora Prof.dr.sc. Goran Martinović

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:

SADRŽAJ

1. UVOD

U sastavu svakog programskog rješenja nalazi se sveobuhvatan niz pozadinskih usluga (engl. *backend services)* namijenjenih za podršku prednjem sučelju (engl. *frontend*) kojeg korisnici vide i koriste. Posao stvaranja cjelokupne pozadinske podrške programskog rješenja nije nimalo jednostavan zadatak, a mnoge organizacije biraju upotrebu postojećih, dokazanih rješenja umjesto stvaranja vlastitih u svrhu uštede resursa i vremena. Bez obzira na domenu programskog rješenja, u većini slučajeva rješenja sadrže sličan skup pozadinskih usluga. Ta tvrdnja se posebno ispunjava u domeni mobilnih i web aplikacija. Upravo zbog navedenog, kao jedan od rastućih trendova u razvoju mobilnih i web aplikacija, ističe se BaaS (engl. *Backend as a Service*). Ideja iza tog trenda je stvoriti jedinstveni sustav koji organizacijama pruža niz pozadinskih usluga koje služe za obradu podataka i podršku prednjem sučelju njihovog specifičnog programskog rješenja. Tema ovog rada odnosi se na izradu vlastitog programskog rješenja za sustav BaaS za web aplikacije. Kao i ostali modeli usluga unutar računarstva u oblaku, model BaaS zasnovan je na konceptu pružanja usluga na zahtjev. Glavni razlozi upotrebe sustava BaaS su povećana brzina i jednostavnost razvoja programskih rješenja. No, nedostaci koji se pojavljuju prilikom upotrebe tuđih sustava su uglavnom posljedica nedostatka kontrole nad tim sustavima. Upravo zato, ovaj diplomski rad argumentira stvaranje vlastitog sustava BaaS, čime se postižu prednosti tog uslužnog modela i eliminiraju nedostaci proizašli iz nedostatka kontrole nad korištenim sustavom.

U drugom poglavlju, prikazane su osnove na kojima počiva sustav BaaS. Predstavljeni su najveći postojeći sustavi BaaS i dana je usporedba prednosti i nedostataka koji se javljaju prilikom korištenja takvim sustavima. U trećem poglavlju, provedena je analiza arhitektura postojećih sustava BaaS, te su izvedeni osnovni arhitekturni koncepti i stilovi na kojima se zasniva dizajn razvijenog sustava. U četvrtom poglavlju, opisuje se postupak izrade vlastitog sustava BaaS. Razvijeni sustav BaaS se temelji na arhitekturi mikrousluga i sastoji se od tri usluge, koje su gotovo neizostavne u današnjim web aplikacijama: pohrana podataka, upravljanje korisnicima i upravljanje datotekama. U petom poglavlju, prikazana je upotreba razvijenog sustava BaaS u razvoju klijentske web aplikacije. Kao posrednik između sustava BaaS i web aplikacije, razvijen je paket za razvoj programa (engl. *SDK - Software Development Kit*). Po završetku razvoja web aplikacije, provedena je analiza čiji je cilj utvrditi udio pozadinskih usluga u web fotogaleriji, kako bi se razjasnilo koliko se ubrzava razvoj web aplikacije i za koliko se smanjuje njena složenost. Posljednje poglavlje odnosi se na zaključak u kojem je dan kratak osvrt na rad te su istaknuti glavni izazovi prilikom razvijanja vlastitog sustava BaaS.

2. OSNOVE SUSTAVA BACKEND AS A SERVICE

U ovom poglavlju, prikazane su osnove na kojima počiva sustav BaaS. Sustav BaaS proizlazi iz novije paradigme informatičke tehnologije koja se naziva računarstvo u oblaku. Filozofija koja proizlazi iz te paradigme je stvaranje raznih uslužnih modela koji se koriste na zahtjev. Osnovni takvi modeli su SaaS (engl. *Software as a Service*), IaaS (engl. *Infrastructure as a Service*) i PaaS (engl. *Platform as a Service*). Iako je sličan standardnim modelima usluga u oblaku, BaaS je jedinstven po specifičnom rješavanju potreba web i mobilnih aplikacija. U poglavlju je također opisana povijest sustava BaaS, koja se pretežno proteže kroz ovo desetljeće. Također su predstavljeni najveći postojeći sustavi BaaS i dana je usporedba prednosti i nedostataka koji se javljaju prilikom korištenja takvim sustavima.

2.1. Pojam i okolina sustava BaaS

BaaS također poznat kao i MBaaS (engl. *Mobile Backend as a Service*) je model povezivanja web i mobilnih aplikacija s pozadinskim uslugama temeljenim na oblaku (engl. *cloud-based*) [1]. Umjesto upotrebe srednjeg sloja (engl. *middleware*), BaaS stvara jedinstveno aplikacijsko sučelje (engl. *API- Application Programming Interface*) i paket za razvoj programa (engl. *SDK - Software Development Kit*) za povezivanje aplikacija s pozadinskim uslugama. BaaS usluge su neke od sljedećih usluga: pohrana podataka, pritisne obavijesti, upravljanje poslužiteljem, upravljanje korisnicima i datotekama, integracija društvenih mreža, usluge lokacije te mnoge druge pozadinske usluge. Usluge sustava BaaS imaju vlastiti API čime je olakšano njihovo integriranje u aplikacije.

Filozofija stvaranja raznih uslužnih modela ima svoj začetak u pojavi računarstva u oblaku (engl. *cloud computing*). Računarstvo u oblaku je paradigma informatičke tehnologije koja omogućava svestran pristup zajedničkim skladištima konfigurabilnih resursa i usluga na višoj razini koje se mogu brzo dobiti pri minimalnom trošku kontrole i najčešće preko Interneta [2]. Pružatelji usluga u oblaku na taj način omogućuju organizacijama da se usredotoče na svoju osnovnu djelatnost umjesto da troše resurse na računalnu infrastrukturu i održavanje. Prema [3], računarstvo u oblaku identificira pet ključnih karakteristika: samoposluživanje na zahtjev, široki pristup mreži, grupiranje resursa, rapidna elastičnost i mjerena usluga.

Pružatelji računalnih usluga u oblaku nude svoje usluge prema različitim modelima, od kojih su tri standardna: Infrastruktura kao usluga (engl. *Infrastructure as a Service - IaaS*), Platforma kao usluga (engl. *Platform as a Service - PaaS*) i Program kao usluga (engl. *Software as a Service -*

SaaS) [3]. Ovi modeli nude različite razine apstrakcije računalnih resursa, pri čemu vrijedi da je mogućnost kontrole nad resursom manja što je razina apstrakcije veća. IaaS se odnosi na sve usluge u oblaku koje pružaju API za razlikovanje detalja niske razine o mrežnoj infrastrukturi, kao što su fizički računalni resursi, lokacija, particije podataka, skaliranje, sigurnost i sigurnosne kopije. PaaS pruža platformu koja korisnicima omogućava razvoj, pokretanje i upravljanje aplikacija čime eliminira potrebu za složenom izgradnjom i održavanjem takve infrastrukture. SaaS je model licenciranja programa i isporuke u kojem je program licenciran na osnovu pretplate i središnje je smješten, a pristupa mu se na zahtjev. Iako je sličan standardnim modelima usluga u oblaku, BaaS je jedinstven po specifičnom rješavanju potreba web i mobilnih aplikacija.

2.2. Povijest sustava BaaS

Prema [4], prvu komercijalnu izvedbu sustava BaaS izradila je tvrtka Firebase 2013. godine. Takvi sustavi su već tada bili u uporabi, primjerice tvrtke Google i Facebook koristile su ih kao interne sustave. No, tvrtka Firebase je prva koja je izložila takvu tehnologiju kao API. Na taj način su omogućili upotrebu njihovog sustava BaaS za opću namjenu. Ideja iza tog postupka je bila zamijeniti tradicionalni model pozadinske podrške, zahtjev - odgovor (engl. *request-response*), s novim modelom koji se temelji na sinkronizaciji podataka. U novom modelu se podaci mogu slati klijentu bez potrebe za njihovim prvotnim zahtjevom. Također je omogućeno zadržavanje stanja na klijentskoj strani. Osim što je eksponiran njihov sustav BaaS za opću upotrebu, tvrtka Firebase je prva uvela opciju plaćanja za usluge sustava. Tako je i za BaaS sustave uveden standardni način plaćanja za usluge računarstva u oblaku, koji sadrži nekoliko tarifa koje se razlikuju po parametrima korištenja (količina podataka, brzina korištenja,…).

Prema [5], najveći i najpoznatiji sustavi BaaS su:

- Firebase, trenutno u vlasništvu tvrtke Google,
- CloudKit od tvrtke Apple,
- Kinvey.

Uz navedene, u istoj kategoriji je bio i sustav Parse od tvrtke Facebook, no on je ugašen 2017. godine. Pružatelji sustava BaaS ostvaruju prihode od korištenja njihovih usluga na različite načine. Prema [6], najčešće se definira ugovor između klijenta i pružatelja sustava BaaS prema kojem klijent prima određeni broj besplatnih aktivnih korisnika ili API zahtjeva mjesečno, a plaća naknadu za svakog korisnika ili poziv, preko tog ograničenja. Alternativno, klijenti mogu platiti određenu naknadu za paket koji omogućuje veći broj zahtjeva ili aktivnih korisnika mjesečno.

Sustavi BaaS se najčešće nude kao komercijalne usluge, ali postoje i opcije otvorenog koda (engl. *open source*) kao što su: BAASBOX, Skygear i Apache Usergrid.

2.3. Usluge sustava BaaS

Domena usluga sustava BaaS proizlazi iz unije domena pozadinskih usluga web i mobilnih aplikacija, s obzirom da je BaaS uslužni model računarstva u oblaku specifičan za usluge takvih vrsta aplikacija. Web aplikacija je računalni program zasnovan na klijent-poslužitelj (engl. *client - server*) modelu, u kojem se klijent pokreće i izvršava u web pregledniku [7]. Mobilna aplikacija je računalni program dizajniran za pokretanje i rad na mobilnom uređaju kao što su mobitel, tablet, pametni sat i drugi.

Web i mobilne aplikacije zahtijevaju sličan skup pozadinskih usluga. Ne postoji striktna definicija koje usluge jedan sustav BaaS mora ili ne smije sadržavati. Pružateljima sustava BaaS je u cilju stvoriti efikasan i efektivan skup usluga, tj. takav skup usluga koji zadovoljava potrebe korisnika i istovremeno ostvaruje veće prihode od troškova njegove izrade. Analizom postojećih sustava BaaS, koji prema [5] imaju najveći udio korisnika na tržištu sustava BaaS, mogu se izdvojiti zajedničke usluge tih sustava. U ovom radu definiraju se standardne usluge sustava BaaS kao zajedničke usluge izdvojene na temelju analize nekoliko najvećih sustava BaaS.

Standardne usluge sustava BaaS su:

- Pritisne obavijesti (engl. *Push notifications*) - poruke koje se pojavljuju na mobilnom uređaju. Izdavači aplikacija mogu ih poslati u bilo kojem trenutku, a korisnici ne moraju upotrebljavati svoje uređaje u tom trenutku da bi ih primili.
- Integracija s društvenim mrežama - rješenje za upravljanje sadržajem svih društvenih mreža pomoću jedne usluge. Takav tip integracije je neophodan za širenje brenda organizacije na popularnim platformama društvenih medija.
- Pohrana podataka - način pohrane lokalno baziranih podataka aplikacije na udaljene sustave za pohranu. Aplikacija koja koristi ovu uslugu može slati svoje podatke na udaljeni sustav za pohranu te obrađivati podatke kroz njega.
- Lokacijske usluge - važna usluga za kontekstualizaciju, analiziranje trendova i ponašanja korisnika. Usluga omogućuje snimanje podataka o lokaciji uređaja koji koriste aplikaciju, kako bi ona učinkovitije ciljala kampanju, pritisne obavijesti i ponude.

- Upravljanje s korisnicima - usluga koja osigurava ovjeravanje autentičnosti i autorizaciju korisnika aplikacije. Omogućava aplikaciji postavljanje korisničkih uloga, grupiranje korisnika u razne grupe te povezivanje s vanjskim sustavima za ovjeravanje autentičnosti.
- Upravljanje datotekama - usluga koja sadrži namjenski prostor za pohranu datoteka. Omogućava aplikaciji učitavanje i preuzimanje datoteka, upravljanje mapama i direktorijima za smještaj datoteka i postavljanje korisničkih prava na operacije nad datotekama.

2.4. Prednosti i nedostaci sustava BaaS

Prema [8], previđa se da će tržište sustava BaaS biti vrijedno više od 28 milijardi dolara do 2020. godine te da će broj razvojnih programera biti veći od 25 milijuna. Dakle, BaaS uslužni modeli postaju jedan od gorućih tehnoloških tržišta te podržavaju najbrže rastući profesionalni segment u svijetu - informatičku tehnologiju. Po predviđenoj raspodjeli razvojnih programera do 2020. godine, programeri će se raspodijeliti po oko 140 tisuća startupa, 230 tisuća programskih razvojnih agencija i nekoliko drugih oblika udruženja. Sve više programera i tvrtki shvaća i prihvaća razloge upotrebe sustava BaaS, što rezultira povećanjem broja usluga koje BaaS pružatelji nude.

Prema [9], razlozi upotrebe sustava BaaS su sljedeći:

- Trošak razvoja - jedna od glavnih svrha sustava BaaS je automatiziranje ponavljajućih zadataka i izbjegavanje dodjeljivanja aktivnosti niske vrijednosti razvojnim programerima. Na taj način, smanjuje se ukupno vrijeme razvoja programskog rješenja i time su ukupni troškovi projekta manji.
- Brzina - ovisno o vrsti pozadinske tehnologije, aplikacija se može ubrzati nekoliko puta. Također omogućuje velikim tvrtkama da se bolje brže prilagode promjenama na tržištu i startupima kako bi brže pripremili svoj minimalni održivi proizvod (engl. *minimum viable product*).
- Jednostavnost upotrebe - krivulja učenja za upotrebu sustava BaaS je obično vrlo niska i zahtijeva vrlo malo napora razvojnog programera. Za razvojne programere, upotreba takvog sustava znači, da može samostalno izraditi cijelo programsko rješenje bez potrebe za razvojem pozadinske podrške.

Kao i svaki uslužni model računarstva u oblaku, sustavi BaaS također imaju svoje prednosti i nedostatke. Definirajući prednosti i nedostatke takvih sustava, ne uzimaju se u obzir tehnologije

izrade, arhitektura ili usluge razvijenog sustava. Svaki pružatelj sustava BaaS koristi različite arhitekture i tehnologije za razvoj te nudi različite usluge svojim korisnicima. Prednosti proizlaze iz samih razloga upotrebe, a nedostaci se definiraju u usporedbi s drugim uslužnim modelima računarstva u oblaku. Također, neki nedostaci sustava BaaS proizlaze iz nedostataka koncepta računarstva u oblaku i zajednički su svim uslužnim modelima te vrste računarstva.

Prema [9], prednosti koje se ostvaruju upotrebom sustava BaaS prilikom razvijanja programskih rješenja su:

- Različitost usluga - tržište sustava BaaS je veliko i ubrzano raste te postoji velik broj pružatelja sustava koji pokrivaju sve osnovne pozadinske usluge programskih rješenja.
- Smanjen napor razvoja - eliminiraju se ponavljajuće, monotone aktivnosti razvojnih programera, čime se smanjuje i vrijeme izrade programskog rješenja a posljedično i ukupni troškovi razvoja.
- Unovčavanje - s obzirom da su napor i vrijeme za razvoj programskog rješenja manji, proizvod se može lansirati prije i početi ostvarivati prihod ranije.
- Fokus na razvoj prednjeg sučelja - programeri se mogu fokusirati na razvoj prednjeg sučelja i prilagoditi ga potrebama tržišta na brz način.
- Skalabilnost - ukupan broj korisnika može narasti vrlo brzo bez zastoja ili smanjenja performansi.
- Sigurnost - većina pružatelja sustava BaaS koristi dobre sigurnosne protokole.

Prema [9], nedostaci upotrebe sustava BaaS prilikom razvoja programskih rješenja su:

- Smanjena kontrola - razvojni programeri obično vole potpunu kontrolu nad izvornim kodom, a većina sustava BaaS ograničava pristup izvornom kodu pozadinskih usluga.
- Blokada pružatelja sustava BaaS - korisnik mora pažljivo pročitati uvjete upotrebe svakog sustava BaaS i procijeniti postoji li prijetnja zaključavanja podataka na strani pružatelja usluge i nemogućnost prijenosa podataka na drugi izvor ako je potrebno.
- Velika ovisnost o pružatelju sustava BaaS - problemi pružatelja ili gašenje sustava BaaS rezultiraju propadanjem svih programskih rješenja stvorenih koristeći usluge tog sustava, pa je potrebno samostalno razviti korištene usluge ili promijeniti pružatelja usluga da bi rješenja ostala funkcionalna.

Povećavanjem broja usluga ili aplikacija koje koriste usluge nekog pružatelja sustava BaaS, sustavi BaaS postaju ključan poslovni resurs za uspješnost misije poduzeća [8]. Kako bi se iskoristile sve prednosti sustava BaaS, a istovremeno izbjegli nedostaci, potrebno je steći vlasništvo nad sustavom BaaS. Vlasništvo sustava BaaS je potrebno kako bi se izbjeglo isključivanje sustava, blokada pružatelja sustava i druge nepovoljne promjene. Nekoliko vodećih pružatelja je ugasilo svoje BaaS sustave tijekom proteklih godina. Najzvučniji primjeri su Parse sustav od tvrtke Facebook, StackMob sustav od tvrtke PayPal i Proxomo sustav od tvrtke Lucent [8]. No, trendovi na tržištu sustava BaaS ukazuju da su prednosti tih sustava brojnije i jače od nedostataka pa korištenje sustavom BaaS više nije upitno, nego se postavlja pitanje koji sustav BaaS koristiti. U ovom radu opisan je postupak izrade vlastitog sustava BaaS čime se postižu prednosti tog uslužnog modela i eliminiraju nedostaci proizašli iz nedostatka kontrole nad korištenim sustavom.

3. ANALIZA I DIZAJN ARHITEKTURE SUSTAVA BaaS

U ovom poglavlju, provedena je analiza arhitekturnih koncepata i stilova potrebnih za dizajn arhitekture sustava BaaS. Pri tome je provedena usporedba najkorištenijih, postojećih sustava BaaS promatrajući blokovske prikaze njihovih arhitektura. Na temelju provedene usporedbe, izvedeni su osnovni arhitekturni koncepti i stilovi koji se koriste prilikom izrade sustava BaaS u ovom radu. Na kraju poglavlja, ostvaren je dizajn arhitekture sustava BaaS, razvijenog u ovom radu, te je arhitektura prikazana u obliku blokovskog prikaza.

3.1. Opis arhitekturnih koncepata, stilova i pristupa

3.1.1. Arhitektura mikrousluga

Arhitektura mikrousluga je jedinstvena metoda razvoja programskih sustava kojoj je posljednjih godina porasla popularnost prilikom odabira arhitektura za poslovne aplikacije [10]. Iako ne postoji standardna, formalna definicija mikrousluga, postoje određene karakteristike koje nam pomažu u prepoznavanju stila. U suštini, arhitektura mikrousluga je metoda razvoja programskih sustava kao paketa samostalno raspoloživih, malih, modularnih usluga u kojima svaka usluga pokreće svoj proces i komunicira kroz dobro definiran i lagan mehanizam koji služi poslovnom cilju programskog sustava. Kako usluge međusobno komuniciraju, ovisi o zahtjevima programskog sustava. U ovom radu, usluge koriste arhitekturni stil REST (engl. *REpresentational State Transfer*) i format podataka JSON.

Baš kao što ne postoji formalna definicija mikrousluga, ne postoji ni standardni model koji će biti reprezentiran u svakom sustavu temeljenom na ovom arhitekturnom stilu. No, prema [10], većina mikrousluga dijeli nekoliko značajnih karakteristika:

- Programski sustav razvijen na ovoj arhitekturi može biti razdvojen u više mikrousluga - mikrousluge se mogu implementirani, modificirati i objaviti bez ugrožavanja integriteta sustava.
- Stil arhitekture mikrousluga omogućuje jednostavnije organiziranje programskog sustava oko poslovnih ciljeva - za razliku od tradicionalnog, monolitnog, arhitekturnog stila, timovi se sastavljaju s obzirom na usluge koje se razvijaju. Tako su odgovornosti svakog tima stvaranje određenih proizvoda na temelju jedne ili više pojedinačnih usluga koje međusobno komuniciraju preko sabirnice poruka.

- Rad mikrousluge odgovara radu poslužitelja - mikrousluge primaju zahtjev, procesiraju ga i generiraju odgovor. Taj mehanizam rada je suprotan radu monolitnih aplikacija, gdje se koriste napredni mehanizmi koordinacije i usmjeravanja.
- Decentralizirano upravljanje i pohrana - s obzirom da se mikrousluge za isti programski sustav razvijaju pomoću različitih tehnologija i alata, provodi se i decentralizirano upravljanje po kojem je za svaku mikrouslugu odgovoran tim koji ju je razvio. Također se koriste decentralizirani sustavi za pohranu podataka, gdje svaka mikrousluga obično upravlja jedinstvenom bazom podataka.
- Jednostavnija detekcija grešaka - s obzirom da se programski sustav sastoji od nekoliko mikrousluga, jednostavnije je prepoznati kojoj mikrousluzi pripada detektirana pogreška.
- Povećana raznovrsnost klijentskih uređaja - jednostavnije je prilagoditi mikrouslugu za rad na drugoj vrsti klijentskog uređaja, nego razviti više monolitnih programskih sustava za različite uređaje.

3.1.2. Arhitekturni stil REST

REST je arhitekturni stil za pružanje standarda između računalnih sustava na internetu, što olakšava njihovu međusobnu komunikaciju [11]. REST kompatibilni sustavi se često nazivaju i *RESTful* sustavi. REST definira skup ograničenja i svojstava temeljenih na protokolu HTTP (engl. *HyperText Transfer Protocol*). Internet usluge koje su u skladu sa arhitekturnim stilom REST osiguravaju interoperabilnost između računalnih sustava na internetu. Takve usluge omogućuju sustavima pristup i manipulaciju resursima na internetu, korištenjem jedinstvene i unaprijed definirane skupine operacije bez stanja (engl. *stateless*).

Prema [12], pojam REST uveo je i definirao Roy Fielding 2000. godine u svojoj doktorskoj disertaciji. Korištenjem standardnim operacijama i protokolima bez stanja, REST sustavi ciljaju na brzu izvedbu, pouzdanost i sposobnost širenja ponovnom uporabom komponenata koje se mogu upravljati i ažurirati bez utjecaja na cijeli sustav. U današnjem volumenu interneta, resursom na internetu se smatra sve ono što se može identificirati, imenovati, adresirati ili obrađivati na bilo koji način. U Fieldingovoj disertaciji se osim stila REST, predstavlja i standard za identifikaciju resursa na internetu, URI (engl. *Uniform Resource Identifier*). Svaki resurs posjeduje vlastiti URI, tj. niz znakova dizajniran za jednoznačno prepoznavanje resursa. U REST sustavima, zahtjev za resursom se izvodi tako da se adresira na njegov URI, a odgovor se generira u nekim od formata za prikaz podataka, pri čemu su najčešći: HTML (engl. *HyperText Markup Language*), XML i

JSON [11]. Odgovor može potvrditi da je došlo do nekih izmjena pohranjenog resursa, a može i pružiti hipertekstualne veze na druge srodne resurse ili zbirke resursa.

Arhitekturni stil REST se bazira na protokolu HTTP, a posljedično koristi četiri standardne HTTP operacije za sastavljanje zahtjeva za resursom:

- GET - dohvaćanje specifičnog resursa (preko identifikatora) ili zbirke resursa.
- POST - kreiranje novog resursa.
- PUT - ažuriranje specifičnog resursa.
- DELETE - brisanje specifičnog resursa.

Prema [11], REST zahtjev se općenito sastoji od:

- Naziva HTTP operacije,
- Zaglavlja,
- Putanje prema resursu,
- Opcionalnog tijela poruke koji sadrži podatke.

U zaglavlju zahtjeva, klijent šalje vrstu sadržaja koju može primiti od poslužitelja. Ta vrsta sadržaja se sprema u polje *Accept*. Opcije za vrste sadržaja su definirane standardom MIME (engl. *Multipurpose Internet Mail Extensions*), po kojem se određuje tip i podtip sadržaja. Primjerice, tekstualna datoteka koja sadrži podatke u formatu podataka HTML bi posjedovala *text/html* vrstu sadržaja, a ona koja sadrži podatke u formatu CSS bi posjedovala *text/css* vrstu sadržaja. Ostali MIME tipovi su: *image*, *audio*, *video* i *application*.

REST zahtjevi moraju sadržavati putanju do resursa na kojem bi operacija trebala biti izvršena. Konvencionalno, prvi dio putanje treba imati oblik množine resursa. To zadržava ugniježđene putanje jednostavnim za čitanje i lako razumljivim. Putanje bi trebale sadržavati informacije potrebne za lociranje resursa s potrebnim stupnjem specifičnosti. Ako se pokušava pristupiti jednom resursu, u putanju se mora dodati identifikator tog resursa. Kada se putanja odnosi na zbirku resursa, nepotrebno je dodavanje identifikatora. Primjerice, ako od zamišljenog sustava fakulteta želimo podatke o specifičnom profesoru na specifičnom kolegiju, pri čemu je identifikator kolegija jednak 37, a identifikator profesora 12, putanja do željenog podatka, po konvenciji, može izgledati ovako: „domena-fakulteta/kolegiji/37/profesori/12“.

U slučajevima kada poslužitelj klijentu šalje odgovor koji sadrži podatke, mora naznačiti vrstu sadržaja u zaglavlju odgovora. Taj podatak o vrsti sadržaja se sprema u polje *Content-Type* zaglavlja odgovora i upozorava klijenta na vrstu podataka koji se šalju u tijelu odgovora. Te vrste sadržaja su također određene standardom MIME, baš kao i u polju *Accept* zaglavlja zahtjeva. Vrsta sadržaja koju poslužitelj šalje u odgovoru bi trebala odgovarati jednoj od vrsti sadržaja koje je klijent naveo u polju prihvaćanja zahtjeva. Odgovori s poslužitelja također sadrže statusne kodove koji upozoravaju klijenta na informacije o uspjehu operacije. Za svaku od četiri osnovne HTTP operacije, postoji očekivani statusni kod, kojeg bi poslužitelj trebao vratiti u slučaju uspjeha:

- GET - statusni kod 200 (*OK*).
- POST - statusni kod 201 (*CREATED*).
- PUT - statusni kod 200 (*OK*).
- DELETE - statusni kod 204 (*NO CONTENT*).

Osim navedenih, postoji velik broj statusnih kodova za opisivanje različitih ishoda primjena HTTP operacija nad traženim resursom.

3.1.3. Format podataka JSON

Prema [13], JSON je uobičajeni format podataka koji se koristi za asinkronu komunikaciju preglednika i poslužitelja, nastao kao zamjena za format podataka XML u nekim sustavima. Lagan je format za razmjenu podataka, lako je čitljiv ljudima i jednostavan za pisanje. Također, jednostavan je za strojnu analizu i generiranje. Temelji se na podskupu JavaScript jezika za programiranje. JSON je tekstualni format koji je potpuno nezavisan od jezika, ali koristi konvencije koje su poznate programerima obitelji C programskih jezika, kao što su C,C++, C#, Java, JavaScript, Perl, Python i mnogi drugi programski jezici. Ova svojstva čine JSON idealnim jezikom razmjene podataka.

JSON je izgrađen pomoću dvije strukture [14]: zbirke parova naziv - vrijednost i neuređene liste vrijednosti. U različitim jezicima, zbirka parova se ostvaruje kao objekt, zapis, struktura, rječnik, ključni popis ili asocijativni niz. Neuređena lista vrijednosti se u većini jezika ostvaruje kao niz, vektor, popis ili slijed. To su univerzalne strukture podataka i gotovo svi moderni programski jezici ih podržavaju u jednom ili drugom obliku. U formatu JSON, oni poprimaju sljedeće oblike:

- Objekt - neuređen skup parova naziv - vrijednost. Počinje i završava sa vitičastim zagradama. Svaki naziv je popraćen dvotočkom, a parovi su odvojeni zarezom.

- Niz - uređena zbirka vrijednosti. Počinje i završava uglatim zagradama, a vrijednosti su odvojene pomoću zareza.
- Vrijednost - vrijednost može biti znakovni niz, broj, istinit ili lažan iskaz, ništavna vrijednost, objekt ili niz. Vrijednosne strukture moraju biti ugniježđene.

3.1.4. Protokol OAuth2

OAuth2 je autorizacijski protokol koji omogućuje dobivanje ograničenog pristupa korisničkim računima unutar HTTP usluga [15]. Djeluje tako da delegira autorizaciju korisnika na uslugu koja je domaćin korisničkog računa i autorizira strane aplikacije za pristup tom korisničkom računu. OAuth2 omogućuje tokove autorizacije za web, mobilne i stolne aplikacije. OAuth2 definira četiri uloge:

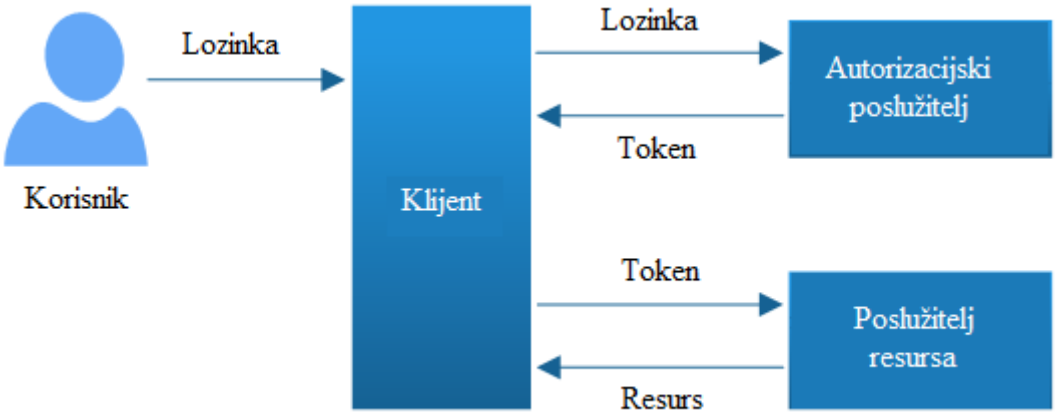
- Vlasnik resursa (engl. *Resource owner*) - aplikacija koja je vlasnik korisničkog računa.
- Poslužitelj resursa (engl. *Resource server*) - poslužitelj koji uslužuje vlasnika resursa.
- Klijent - aplikacija koja zahtijeva pristup poslužitelju resursa.
- Autorizacijski poslužitelj (engl. *Authorization server*) - poslužitelj koji predaje pristupnu značku klijentu.

Autorizacijski poslužitelj i poslužitelj resursa mogu biti isti, što je čest slučaj [15].

Značke (tokeni) su slučajni nizovi generirani od autorizacijskog poslužitelja i izdaju se kada ih klijent zatraži. Postoje dvije vrste znački: pristupna značka (engl. *access token*) i značka za osvježavanje (engl. *resfresh token*). Pristupna značka je najvažnija, jer omogućuje pristup podacima korisnika stranim aplikacijama. Klijent šalje ovu značku, kao parametar zaglavlja REST zahtjeva, prema poslužitelju resursa. Ima ograničen vijek trajanja, kojeg definira autorizacijski poslužitelj. Značka za osvježavanje je izdana zajedno s pristupnom značkom, ali se ne šalje u svakom zahtjevu. Ona samo služi za obnovu pristupne značke, kada je on istekao.

Protokol OAuth2 ima definirani tok autorizacije s četiri aktivnosti, koji je prikazan na slici 3.1.

1. Korisnik šalje ime i lozinku klijentu.
2. Klijent šalje vjerodajnice (engl. *credentials*) autorizacijskom poslužitelju.
3. Autorizacijski poslužitelj ovjerava autentičnost vjerodajnice i vraća pristupnu značku.
4. Da bi pristupio zaštićenom resursu, klijent mora postaviti pristupnu značku u polje *Authorization* u zaglavlju HTTP zahtjeva.



Sl. 3.1. Tok autorizacije definiran u protokolu OAuth2 [33]

3.1.5. Pristup API First

Uzimajući u obzir da je općeniti sustav BaaS jedan oblik uslužnih modela računarstva u oblaku, domena programskog rješenja razvijenog u ovom radu je također unutar računarstva u oblaku. Prilikom razvijanja programskih sustava u oblaku, *API First* je temeljni princip [16]. Ukoliko je krajnji cilj da programski sustav bude sudionik ekosustava usluga, u tom slučaju je potrebno primijeniti koncept pristupa *API First*. Pristup *API First* je razvojna strategija u kojoj je prvi poslovni prioritet razviti API koji najprije definira interese razvojnih programera, a zatim se programski proizvod gradi na temelju razvijenog API-ja. Izgradnjom na temelju API-ja, programeri štede puno posla prilikom postavljanja temelja za ostale programske proizvode.

U programskom inženjerstvu, API ili jedinstveno aplikacijsko sučelje, je skup definicija, protokola i alata za izradu programskih sustava [17]. Općenito, to je skup jasno definiranih metoda komunikacije između različitih programskih komponenti. Dobro definiran API olakšava razvoj programskih sustava, pružanjem svih temeljnih blokova koje programer zatim sastavlja. API može biti napisan za web sustav, operativni sustav, sustav baze podataka, računalni hardver ili biblioteku programa. API specifikacija može imati mnoge oblike, ali često sadrži specifikacije za rutine, strukture podataka, klase objekata, varijable ili udaljene pozive. Obično se osigurava dokumentacija za API kako bi se olakšala upotreba i implementacija.

Prema [16], postoje tri principa pristupa *API First*:

1. API je prvo korisničko sučelje razvijanog programa - to znači da su programeri koji razvijaju na temelju definiranog API-ja, zapravo korisnici API-ja, pa prema tome API mora biti prikladno definiran. API je način na koji programski proizvod izlaže svoju

funkcionalnost, pa ako u njemu nisu obuhvaćene sve funkcionalnosti proizvoda, njih ne može pokriti ni grafičko korisničko sučelje. Budući da je API prvi i najvažniji način pristupa i interakcije sa programskim proizvodom, on mora biti upravljan i planski razvijan. Potrebno je uložiti vrijeme i trud u dizajniranje API-ja, kao što se ulaže u razvoj korisničkog sučelja.

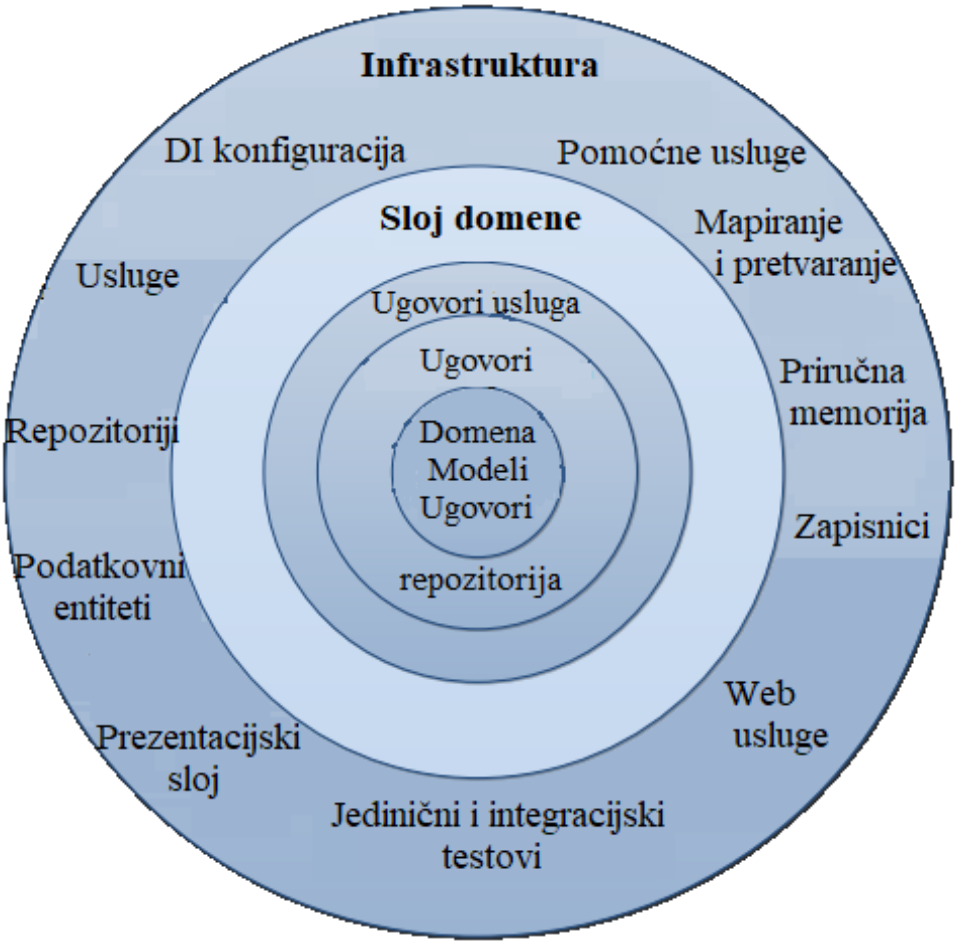
2. API dolazi prije implementacije - implementacija se mijenja kako se aplikacija razvija i programeri optimiziraju, modificiraju i povećavaju funkcionalnosti. Međutim, API se ne bi trebao često mijenjati, nego bi trebao polako i upravljano rasti. Važno je razmotriti rast API-ja u smislu povećanja fleksibilnosti. Obrađivanje API-ja, kao neovisnog o implementaciji, omogućuje razdvajanje API-ja i implementacije. Time API postaje ugovor i specifikacija za implementaciju.
3. API mora biti opisan (ili samo-opisan) - da bi se API mogao koristiti, moraju ga razumjeti osobe koje nisu bile uključene u njegovo stvaranje, što podrazumijeva pisanje dokumentacije. Iskoristiva API dokumentacija je neophodan preduvjet za njegovu upotrebu. Kada je u pitanju dokumentacija za API, strukturirana dokumentacija nadvladava nestrukturiranu. Slijedeći standardne obrasce za URI, vrste resursa, metode zahtjeva, zaglavlja, parametre zahtjeva i oblike odgovora, olakšati će se istraživanje i razumijevanje funkcionalnosti API-ja. Uz opisane API-je, samo-opisni API-ji čine još bolju podlogu za njihovu upotrebu. Samo-opisni API-ji su API-ji opisani pomoći hipermedijskih konstrukata poput veza koje omogućuju otkrivanje drugih resursa API-ja. Na taj način se stvara zatvorena petlja od dizajna, preko implementacije sve do dokumentacije.

3.1.6. Višeslojna arhitektura

U programskom inženjerstvu, višeslojna arhitektura (engl. *multilayered architecture*) je klijent - poslužitelj arhitektura u kojoj se prezentacija, obrada podataka i funkcije upravljanja podacima fizički odvajaju. Najraširenija upotreba višeslojne arhitekture je troslojna arhitektura (engl. *three-tier architecture*) [18]. Višeslojna arhitektura pruža model pomoću kojeg programeri mogu stvoriti fleksibilna programska rješenja za višekratnu uporabu. Segmentiranjem programa u slojeve, razvojni programeri stječu mogućnost mijenjanja ili dodavanja određenog sloja umjesto obrade cijelog programa.

Prilikom razvijanja sustava BaaS u ovom radu, primijenjena je specifična višeslojna arhitektura *Onion*. Na temelju te arhitekture dizajnirane su usluge koje nudi sustav BaaS, a implementirane su u obliku API-ja. Višeslojna arhitektura *Onion* je arhitektura koja ima slojeve definirane od jezgre do infrastrukture i programskog koda. Grafički prikaz te arhitekture, na slici 3.2, podsjeća na slojeve luka, pa otuda dobiva naziv. Slojevi arhitekture mogu ovisiti samo o unutarnjim slojevima, ali nemaju pojam o vanjskim slojevima. Programski sustav temeljen na višeslojnoj arhitekturi *Onion* sadrži pet slojeva:

1. Sloj domene (engl. *Domain layer*) - u ovom sloju se nalaze dijelovi programskog sustava koji su relevantni za domenu problema kojeg taj sustav rješava. To uključuje klase koje predstavljaju poslovne modele te sučelja koja definiraju poslovne operacije i funkcije.
2. Sloj pristupa podacima (engl. *Data access layer*) - omogućuje pojednostavljeni pristup podacima pohranjenim u nekoj vrsti pohrane, najčešće u bazi podataka. Tijesno je vezan uz tehnologiju baze podataka, te vraća referencu na bazu i njene strukture koju potom koristi gornji sloj repozitorija.
3. Sloj repozitorija (engl. *Repository layer*) - pruža apstrakciju podataka u bazi, tako da gornji slojevi koriste tu apstrakciju koja ima definirano sučelje, približno jednako funkcijama baze podataka. Dodavanje, uklanjanje, ažuriranje i odabir stavki iz repozitorija se obavlja kroz niz jednostavnih metoda bez potrebnog znanja o bazi podataka u kojoj se podaci nalaze te njezinim mogućnostima.
4. Sloj usluga (engl. *Service layer*) - nalazi se između sloja repozitorija i aplikacijskog sloja. Zadužen je za upravljanje poslovnim pravilima pretvaranja i prevođenja podataka između ta dva sloja. Također pruža apstrakciju, ali poslovne logike i centralizira vanjski pristup podacima, tj. sloju repozitorija.
5. Aplikacijski sloj (engl. *Application layer*) - struktura ovog sloja ovisi o vrsti programskog sustava koji se razvija. Zadužen je za primanje podataka od klijenta, formatiranje primljenih podataka, te prosljeđivanje nižim slojevima novostvoreni format na obradu. Također obavještava klijenta o rezultatima operacije nižih slojeva i primjenjuje određena validacijska pravila na unesene podatke klijenta. Uglavnom služi za komunikaciju s klijentom, neovisno o tome je li taj sloj API, web aplikacija, mobilna aplikacija ili neki drugi programski oblik.



Sl. 3.2. Složeni dijagram višeslojne arhitekture *Onion* [19]

Osnovno načelo višeslojne arhitekture *Onion* je da je model domene u sredini arhitekture, a vanjski slojevi mogu komunicirati samo s unutarnjim slojevima i to s više njih. Ovakva struktura je složenija od standardne višeslojne arhitekture, ali omogućuje veću fleksibilnost, održivost i skalabilnost. Slojevi su lako razdvojivi, a međusobno komuniciraju preko sučelja. Upravo zato što vanjski sloj koristi samo sučelje za komunikaciju s unutarnjim slojem, mijenjanje implementacije unutarnjeg sloja nimalo ne utječe na rad vanjskog sloja, ukoliko je sučelje nepromijenjeno. Neke prednosti ovakve arhitekture su sljedeće [19]:

- Više programskih sustava može ponovno iskoristiti iste komponente,
- Omogućuje razvojnim timovima da rade na različitim dijelovima programskog sustava,
- Omogućuje razvijanje labavo spojenih sustava,
- Različite komponente sustava mogu biti neovisno razvijane i održavane,
- Moguće je testiranje komponenti neovisne jedna o drugoj.

Međutim, ovaj tip višeslojne arhitekture ima određene nedostatke:

- Zahtijeva duži implementacijski period,
- Može imati negativan utjecaj na performanse,
- Teži postati složena,
- Dodaje nepotrebnu složenost programskom sustavu.

Odabir neke arhitekture se ostvaruje u slučajevima kada su posljedice prednosti arhitekture značajnije od posljedica nedostataka. Prema [19], višeslojna arhitektura *Onion* se upotrebljava u slučaju projekata srednje veličine, koji se brzo mijenjaju a na njima sudjeluju veliki timovi. Kako bi se ipak iskoristile glavne prednosti ove arhitekture, potrebno je primijeniti određene principe na kojima je ona zasnovana. Višeslojna arhitektura *Onion* je zasnovana na načelu ubrizgavanja ovisnosti (engl. *Dependency Injection*) i načelu inverzije kontrole (engl. *Inversion of Control*).

3.1.7. Načelo ubrizgavanja ovisnosti

Ubrizgavanje ovisnosti je načelo po kojem jedan objekt ili statička metoda osigurava ovisnost drugog objekta. Ovisnost je usluga koja se može koristiti [20]. Ubrizgavanje znači donošenje usluge klijentu koji će ju koristiti te je ona dio stanja klijenta. Prebacivanje usluge klijentu, umjesto da klijent gradi ili pronalazi uslugu, temeljni je zahtjev načela ubrizgavanja ovisnosti. Ovaj temeljni zahtjev znači da je korištenje uslugama proizvedenim iz novih klasa ili statičkih metoda zabranjeno. Klijent treba prihvatiti usluge izvana, a tako se može usredotočiti na stvaranje njemu svojstvenih usluga.

Postoje tri uobičajena načina ostvarivanja načela ubrizgavanja ovisnosti: ubrizgavanja na bazi članova klase (engl. *setter injection*), ubrizgavanje na bazi sučelja (engl. *interface injection*) i ubrizgavanje na razini konstruktora (*engl. constructor injection*) [20]. Za primjenu načela ubrizgavanja ovisnosti u svrhu dizajniranja višeslojne arhitekture, potrebno je slijediti sljedeća pravila [19]:

- Zajedničke klase (engl. *common classes*) ne ovise o ničemu,
- Klase domene ovise o zajedničkim klasama i međusobno jedna o drugoj,
- Klase usluga ovise samo o klasama domena,
- Klase infrastrukture o klasama usluga i o klasama domena.

Uporabom načela ubrizgavanja ovisnosti u dizajniranju višeslojne arhitekture, olakšano je jedinično testiranje (engl. *Unit Testing*) te zamjena slojeva pomoću inverzije kontrole.

3.1.8. Načelo inverzije kontrole

U programskom inženjerstvu, inverzija kontrole je načelo dizajna programskih sustava, u kojem razvijani dijelovi programa dobivaju protok kontrole iz općeg razvojnog okvira (engl. *framework*). Arhitekture programskih sustava s ovim načelom dizajna, invertiraju kontrolu u odnosu na tradicionalno proceduralno programiranje. U tradicionalnom programiranju, prilagođeni kod (engl. *custom code*) koji izražava svrhu programa, poziva biblioteke koje se mogu koristiti više puta za generičke zadatke. No, s inverzijom kontrole, razvojni okvir biva zadužen za poziv prilagođenog dijela koda, tj. koda specifičnog za određeni zadatak [21].

Načelo inverzije kontrole se koristi za povećanje modularnosti programa i proširivanje, te ima primjene u objektno orijentiranom programiranju i drugim programskim paradigmama. Taj je pojam prvi upotrijebio Michael Mattsson, a popularizirali su ga Stefano Mazzocchi, Robert C. Martin i Martin Fowler. Pojam se povezuje, ali se razlikuje, od načela ubrizgavanja ovisnosti. Prilikom dizajniranja višeslojne arhitekture, ovo načelo se ostvaruje zamjenom jedne apstrakcije s drugom ili zamjenom jednog implementacijskog sloja s drugim. Ključne značajke koje donosi primjena ovog načela su: razdvajanje (engl. *decoupling*), povećana modularnost i povećana proširivost [19].

3.2. Usporedba arhitektura postojećih sustava BaaS

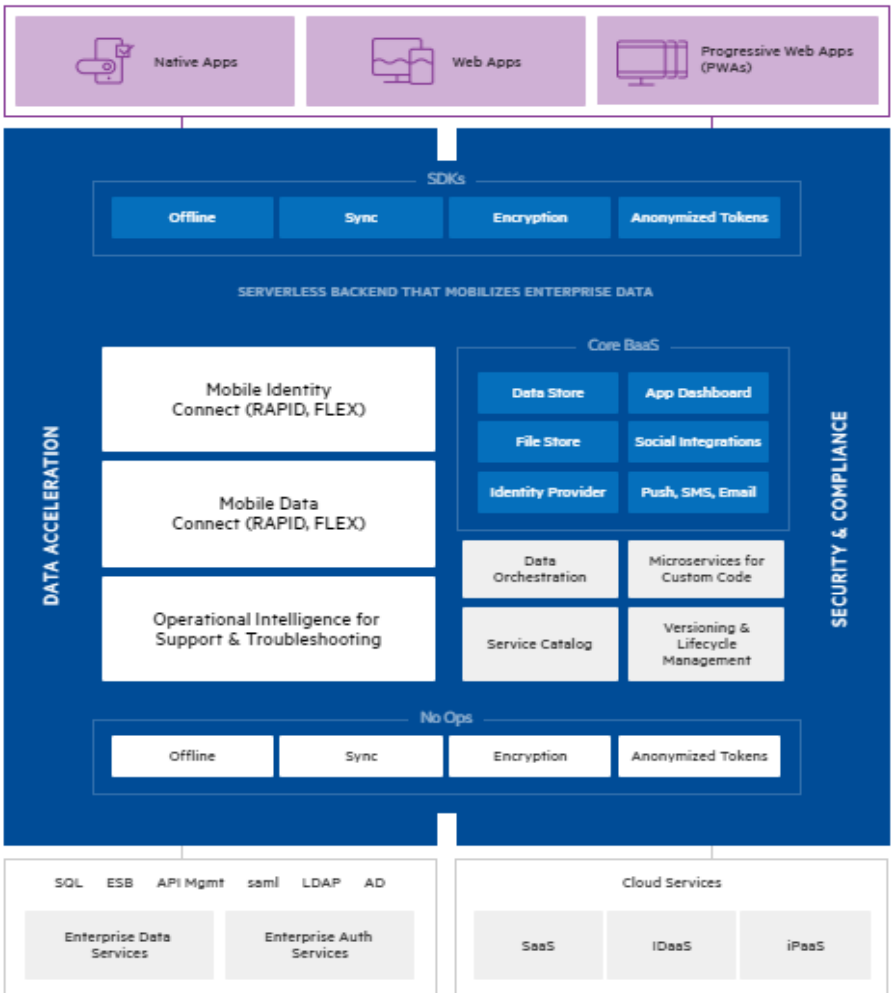
Dobro osmišljena arhitektura potiče razvojne programere na njenu upotrebu i dijeljenje s drugima, kreirajući čvrsti ciklus gdje svaka dodatna uspješna implementacija dovodi do veće angažiranosti i većeg doprinosa razvojnih programera u stvaranju dodane vrijednosti programskom rješenju. Da bi se postigao taj cilj, arhitektura i metodologija usredotočuju se na [8]:

- Jednostavnost - konzistentna i modularna arhitektura,
- Fleksibilnost - brza promjena i dodavanje novih usluga,
- Proširivost - jednostavno dodavanje novih usluga bez utjecaja na druge usluge i platformu.

Analizom postojećih sustava BaaS mogu se utvrditi sličnosti i razlike njihovih arhitektura. Za usporedbu arhitektura postojećih sustava BaaS, odabrani su sustavi Parse, Kinvey i Backendless.

Na slici 3.3 prikazan je blokovski dijagram arhitekture sustava Kinvey. Poopćeno arhitekturu možemo podijeliti na tri dijela: sustav BaaS koji se sastoji od tri podsustava, klijentski dio i dio uslužnih modela i pohrane podataka. Jezgreni dio sustava BaaS se sastoji od usluga: pohrana

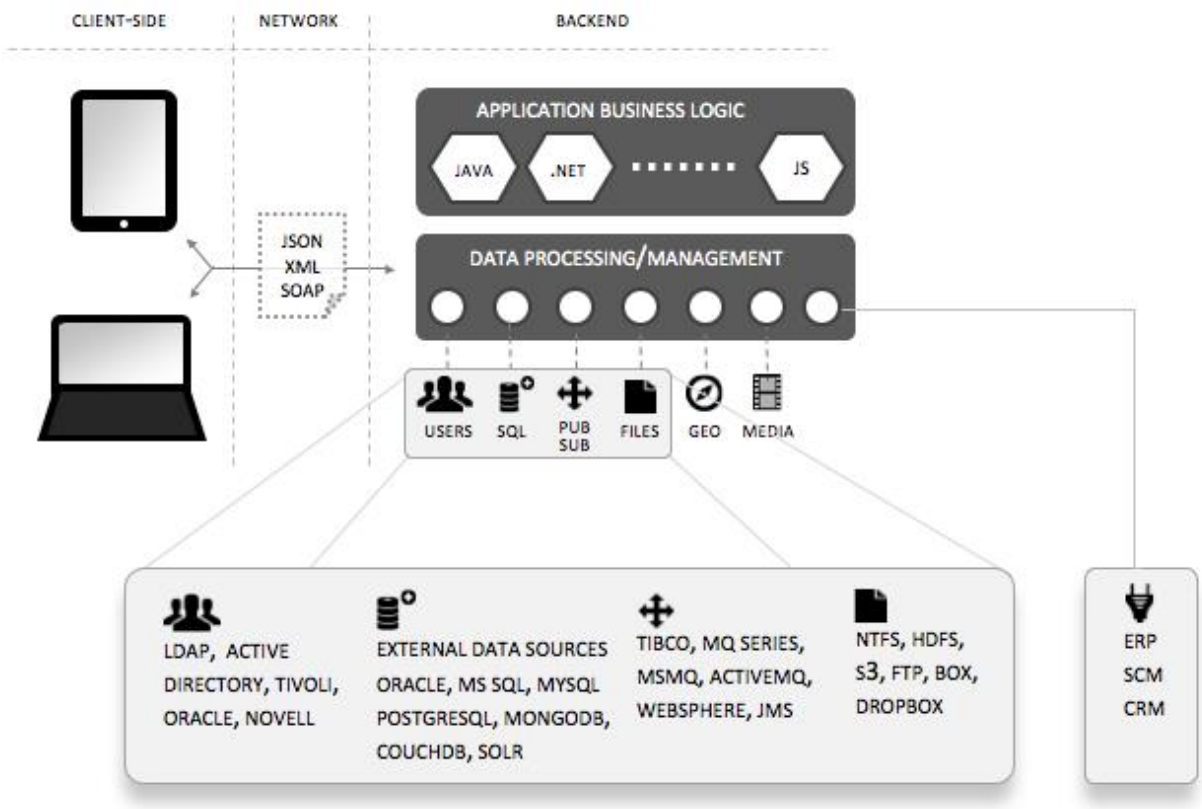
podataka, pohrana datoteka, pružatelj identiteta, kontrolna ploča aplikacije, integracija sa društvenim mrežama i pritisne obavijesti i komunikacija. Ispod sloja usluga dodani su podsustavi za sigurnost i usklađenost. Paralelno s tim podsustavom unutar blokovskog dijagrama, nalazi se podsustav za ubrzanje toka podataka. U njemu su različita programska rješenja i tehnologije kojima se nastoji ubrzati tok podataka između sustava BaaS i klijentskog dijela, ali i pohrane podataka. Sučelje prema klijentskom dijelu se sastoji od grupe SDK-ova za različite klijentske tehnologije. U donjem dijelu blokovskog dijagrama, nalaze se podsustavi za pohranu podataka i podsustav za ostale uslužne modele. Sustav Kinvey koristi različite tehnologije baza podataka prvenstveno zbog različitih potreba usluga koje nudi jezgreni dio. Ostali uslužni modeli su potrebni za implementiranje infrastrukture i platforme na kojima se sustav BaaS implementira, pokreće i upravlja.



Sl. 3.3. Blokovski dijagram arhitekture sustava Kinvey [22]

Na slici 3.4 prikazan je blokovski dijagram arhitekture sustava Backendless. Kod ovog sustava BaaS, jasno se mogu razaznati tri glavna dijela: pozadinska podrška, mrežni dio i klijentske aplikacije. Kao i kod sustava Kinvey, jezgreni dio sustava BaaS čine njegove usluge: pohrana

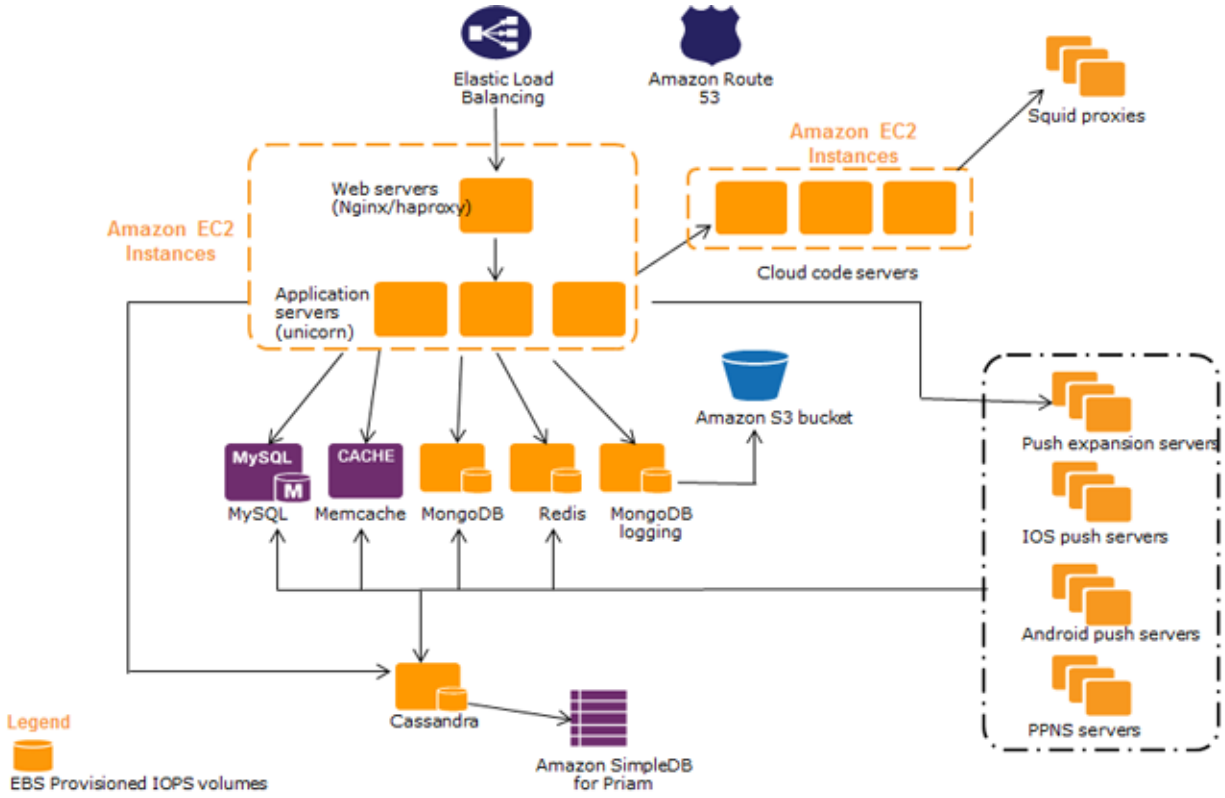
podataka, pohrana datoteka, upravljanje korisnicima, komunikacija i geolokacijske usluge. Svaka od navedenih usluga može biti razvijena različitom tehnologijom. Također usluge koriste različite tehnologije baza podataka i uslužne modele za implementaciju i rad s podacima. U sustavu Backendless, klijentske aplikacije komuniciraju s uslugama sustava BaaS predstavljajući podatke u tri različita formata: JSON (engl. *JavaScript Object Notation*), XML (*engl. EXtensible Markup Language*) i SOAP (engl. *Simple Object Access Protocol*).



Sl. 3.4. Blokovski dijagram arhitekture sustava Backendless [23]

Na slici 3.5 prikazan je blokovski dijagram arhitekture sustava Parse. Iako je preuzet pod vlasništvo tvrtke Facebook, Parse je razvijen na arhitekturi AWS (engl. *Amazon Web Services*). AWS je također sustav implementiran po konceptu uslužnih modela računarstva u oblaku, u vlasništvu tvrtke Amazon, pri čemu se ne može poistovjetiti s jednim standardnim uslužnim modelom nego se smatra kao njihova kombinacija. Arhitektura sustava Parse je konceptualno drugačija od arhitektura sustava Kinvey i Backendless. Slika 3.5 ne prikazuje pozadinske usluge koje sustav Parse nudi, a koje su bile sastav jezgrenih dijelova sustava Kinvey i Backendless. Na slici je prikazan samo dio sustava BaaS koji služi za pohranu podataka i uporabu drugih uslužnih modela računarstva u oblaku. Primjetno je da se koristi kombinacija relacijskih baza podataka, kao što je MySQL i nerelacijskih baza podataka (engl. *NoSQL*), kao što su Redis i MongoDB. Za

potrebe zapisivanja aktivnosti, uporabe priručne memorije i skaliranja baza podataka koriste se usluge AWS sustava, kao što su Amazon S3 bucket, Amazon EC2 Instances i Amazon SimpleDB.

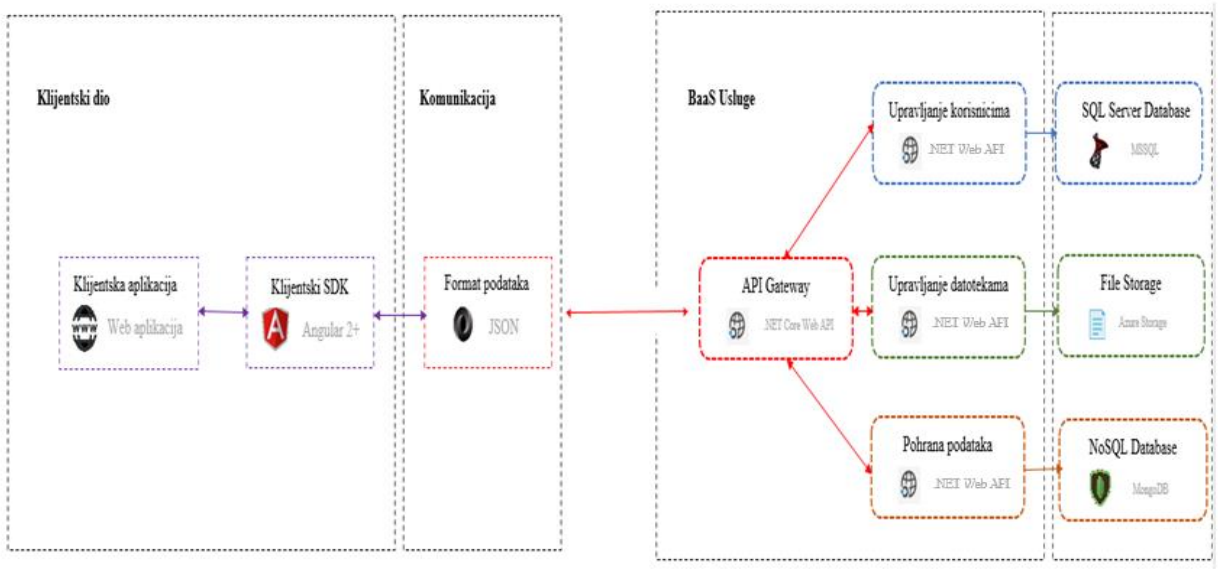


Sl. 3.5. Blokovski dijagram arhitekture sustava Parse [24]

Usporedbom sustava Kinvey, Backendless i Parse mogu se utvrditi njihove razlike, ali i sličnosti na kojima se temelje osnovi zahtjevi arhitekture sustava BaaS koji se razvija u ovom radu. Prve upadljive razlike su u skupu usluga koje nude ti sustavi BaaS. Isto vrijedi i za tehnologije korištene pri razvoju usluga sustava BaaS. Naravno, skup usluga prvenstveno ovisi o korisnicima pružatelja sustava BaaS a tehnologije o osobnim preferencijama razvojnih programera i o zahtjevima usluga. Druga razlika je u složenosti dodatnih podsustava koji uslugama sustava BaaS pridaju veću sigurnost, brzinu, mogućnost skaliranja, sinkronizaciju i enkripciju. Najveća sličnost ovih sustava je koncept rada po kojem je i općeniti sustav BaaS definiran. Sve ove sustave čine tri glavna dijela arhitekture: klijentski dio, usluge sustava BaaS i uslužni modeli računarstva u oblaku. Klijentske aplikacije koriste usluge sustava BaaS, pri čemu mora biti definiran način komunikacije među njima. Usluge sustava BaaS moraju imati definirane formate podataka koje primaju i šalju klijentskim aplikacijama. Unutar BaaS usluga definira se logika rada usluga. Drugi uslužni modeli u oblaku BaaS uslugama služe kao sustavi za pohranu ili kao vanjska potpora u izvođenju njihove namjene.

3.3. Specifikacija arhitekture predloženog programskog rješenja

Arhitektura programskog rješenja u ovom radu, specificirana je prema zaključcima usporedbe arhitektura postojećih sustava BaaS. Prema konceptualnoj podjeli arhitektura promatranih sustava, programsko rješenje u ovom radu se također sastoji od tri dijela: klijentskog dijela, BaaS usluga i dijela za pohranu podataka. Na slici 3.6 prikazan je blokovski dijagram arhitekture razvijenog sustava BaaS.



Sl. 3.6. Blokovski dijagram arhitekture razvijenog sustava BaaS

Temelj arhitekture sustava BaaS leži u konceptima inkrementalnog razvoja heterogenih mikrousluga. Takva arhitektura se naziva arhitektura mikrousluga. Sustav BaaS, razvijen u ovom radu, pruža tri usluge: upravljanje korisnicima, upravljanje datotekama i pohranu podataka. Svaka od navedenih usluga izložena je na upotrebu u obliku API-ja. Također je razvijen jedan API koji predstavlja vrata sustava BaaS (engl. *API Gateway*). Njegovim postojanjem, klijentima se sustav BaaS prikazuje kao jedan API, koji sadrži uniju funkcionalnosti BaaS usluga. No u pozadini, on služi kao raspoređivač zadataka, pri čemu prima zahtjev klijenta, utvrđuje koja usluga mora preuzeti taj zahtjev i prosljeđuje joj zahtjev. Također koordinira pozivanje usluga, čime se povećava skalabilnost, što je jedan od glavnih zahtjeva kvalitete sustava BaaS. API vrata sustava BaaS mogu pozivati više instanci iste usluge, istovremeno ispunjavajući zahtjeve različitih korisnika. Sustav BaaS ima mogućnost registriranja različitih aplikacija i korisnika unutar aplikacija. Za osiguravanje resursa svake aplikacije, koristi se ovjera autentičnosti putem protokola OAuth2. Svaki zahtjev prema BaaS sustavu u zaglavlju mora imati pristupnu značku u polju *Authorization*.

Svaka usluga sustava BaaS u ovom radu, dizajnirana je prema arhitekturnom stilu REST. S uslugama se komunicira putem protokola HTTP, a zahtjevi i odgovori prate standardne stila REST. Za implementaciju arhitekturnog stila REST, koristi se princip *API First*. Web API je jednostavan način za implementaciju REST usluge. Za svaku uslugu, definirani su poslovni zahtjevi u obliku API-ja, te nakon toga slijedi implementacija u odabranoj tehnologiji. Svaki web API, za odgovarajuću BaaS uslugu, dizajniran je po višeslojnoj arhitekturi *Onion* i implementiran koristeći razvojni okvir .NET Web API.

Jedno od načela arhitekture mikrousluga je decentralizirana pohrana i upravljanje podacima. U ovom BaaS sustavu, svaka usluga sadrži vlastiti sustav za pohranu podataka. Osim što su sustavi za pohranu posebni za svaku uslugu, također su izabrane različite tehnologije za pohranu. Usluga pohrane podataka koristi nerelacijsku bazu podataka MongoDB. Na taj način se postiže veća skalabilnost i jednostavnost nego upotrebom relacijskih baza podataka. Upotrebom nerelacijske baze podataka, ostavlja se korisniku prostor za definiranje njene strukture, dok je to upotrebom relacijske baze vrlo teško postići. Svaka klijentska aplikacija koja koristi sustav BaaS, ima mogućnost definiranja shema koje su ekvivalenti modelima aplikacije te mogućnost popunjavanja shema resursima u formatu JSON. Usluga upravljanja datotekama, dodjeljuje namjenski prostor za pohranu datoteka. Za namjenski prostor je odabrana tehnologija Azure Blob Storage, zbog kompatibilnosti s platformom .NET. Usluga upravljanja korisnicima koristi relacijsku bazu podataka MSSQL za pohranu korisnika. Odabrana je relacijska baza podataka zbog mogućnosti definiranja tablice korisnika za svaku aplikaciju. Kao tehnologija za bazu podataka ove usluge, odabrana je baza podataka MSSQL, zbog njene kompatibilnosti s platformom .NET.

Naposljetku, uporaba razvijenog sustava BaaS, unutar web aplikacije, ostvaruje se preko kreiranog SDK-a za razvojni okvir Angular. Angular je razvojni okvir za izradu web aplikacija temeljen na programskom jeziku TypeScript, razvijen od tvrtke Google. Kombinacijom najmodernijih praksi olakšava izradu aplikacija za preglednike, mobilne uređaje ili radne površine. Zbog kompatibilnosti programskih jezika TypeScript i C#, odabran je razvojni okvir Angular za kreiranje SDK-a koji omogućava upotrebu razvijenog sustava BaaS. Angular SDK je razvijen kao Angular modul u koji je smješten kod za komunikaciju s sustavom BaaS. Klijent koji želi koristiti razvijeni SDK mora ga instalirati u vlastitu web aplikaciju kao npm paket i jednostavno pozivati njegove funkcije.

4. PROGRAMSKO RJEŠENJE - SUSTAV BaaS ZA WEB APLIKACIJE

U ovom poglavlju, opisuje se postupak izrade vlastitog sustava BaaS. Prvo je dan prikaz korištenih tehnologija i alata, a nakon toga je opisana struktura svake usluge koju sustav BaaS pruža. Razvijeni sustav BaaS se temelji na arhitekturi mikrousluga i sastoji se od tri usluge, koje su gotovo neizostavne u današnjim web aplikacijama: pohrana podataka, upravljanje korisnicima i upravljanje datotekama. Za svaku od navedenih usluga, dan je prikaz njihove unutarnje arhitekture i funkcionalnosti koje ta usluga omogućuje.

4.1. Opis platformi, alata i tehnologija

4.1.1. Programski jezik C#

C# je elegantan i siguran objektno orijentirani jezik koji programerima omogućuje izradu sigurnih i robusnih aplikacija koje se izvode na platformi .NET [25]. Može se koristiti za izradu Windows klijentskih aplikacija, XML web usluga, raspodijeljenih komponenata, klijent - poslužitelj aplikacija, aplikacija za baze podataka i još mnogo toga. Sintaksa programskog jezika C# je vrlo izražajna, ali je također jednostavna za učenje. Pripada familiji C programskih jezika, pa je prepoznatljiva programerima u programskim jezicima C, C++ ili Java. Programski jezik C# pojednostavljuje mnoge složenosti programskog jezika C++ i pruža moćne značajke kao što su prazne vrijednosti (engl. *null value*), nabrajanja (engl. *enumerations*), delegate, lambda izraze i izravni pristup memoriji. Također podržava generičke metode i tipove koji pružaju povećanu sigurnost i performanse te iteratore koji omogućavaju implementaciju klasa zbirke za definiranje prilagođenih ponašanja iteracija.

Kao objektno orijentirani programski jezik, C# podržava koncepte enkapsulacije, nasljeđivanja i polimorfizma [25]. Sve varijable i metode, uključujući glavnu metodu, ulaznu točku aplikacije, su enkapsulirane unutar definicija klase. Klasa može naslijediti izravno iz jedne roditeljske klase, ali može implementirati bilo koji broj sučelja. Metode koje prepisuju virtualne metode u roditeljskoj klasi zahtijevaju prepisivanje ključne riječi kao način izbjegavanja slučajne redefinicije. U programskom jeziku C#, struktura je poput lagane klase, alocira se kao stog u memoriji, može implementirati sučelja, ali ne podržava nasljeđivanje.

Pored ovih osnovnih objektno orijentiranih načela, C# olakšava razvoj programskih komponenti putem nekoliko inovativnih jezičnih konstrukata, uključujući sljedeće:

- Delegati - enkapsulirani potpisi metoda koji omogućuju sigurne događaje obavještavanja.

- Svojstva (engl. *properties*) - služe kao pomoćne metode za varijable privatnih članova.
- Atributi - daju deklarativne metapodatke o vrstama u vrijeme izvođenja.
- XML komentari - pisani uz programski kod, a čine dokumentaciju programa.
- LINQ (engl. *Language Integrated Query*) - pruža ugrađene sposobnosti upita diljem raznih izvora podataka.

4.1.2. Platforma .NET

C# programi se izvode na platformi .NET. Ona je integralna komponenta Windows operacijskog sustava, a sadrži virtualni sustav izvršavanja koji se naziva CLR (engl. *Common Language Runtime*) sustav i jedinstveni skup biblioteka klasa (engl. *class libraries*) [25]. CLR je komercijalna implementacija CLI (engl. *Common Language Infrastructure*) standarda, od tvrtke Microsoft. CLI je međunarodni standard koji je osnova za stvaranje okruženja za izvršavanje i razvoj u kojima jezici i biblioteke neprimjetno rade zajedno.

Izvorni kod koji je napisan u programskom jeziku C#, prevodi se u posredni jezik (engl. *Intermediate Language-IL*) koji je u skladu s CLI specifikacijom. Kod i resursi posrednog jezika se pohranjuju na disku u izvršnu datoteku zvanu sklop (engl. *assembly*), obično s nastavkom .exe ili .dll. Sklop sadrži manifest koji pruža informacije o vrstama, verziji, kulturi i sigurnosnim zahtjevima.

Kad se C# program izvršava, sklop se učitava u CLR i poduzimaju se određene radnje, ovisno o sadržaju manifesta sklopa. Ako se zadovolje sigurnosni zahtjevi, CLR izvršava JIT (engl. *Just In Time - JIT*) kompilaciju za pretvaranje koda posrednog jezika u izvorne strojne instrukcije. CLR također pruža i druge usluge vezane uz automatsko skupljanje smeća, upravljanje iznimkama i upravljanje resursima. Kod koji CLR izvršava, naziva se i upravljanim kodom, za razliku od neupravljanog koda koji se prevodi u strojni jezik koji cilja određeni sustav.

4.1.3. Razvojno okruženje Visual Studio

Visual Studio je integrirano razvojno okruženje (engl. *Integrated Development Environment- IDE*) razvijeno od tvrtke Microsoft. Koristi se za razvoj računalnih programa, kao i web stranica, web aplikacija, web usluga i mobilnih aplikacija [26]. Visual Studio koristi Microsoftove platforme za razvoj programa kao što su Windows API, Windows Forms, Windows Presentation Foundation, Windows Store i Microsoft Silverlight. Može proizvesti i izvorni i upravljački kod.

Visual Studio sadrži uređivač koda koji podržava komponentu za dovršetak koda, IntelliSense, kao i mogućnost preslagivanja koda (engl. *code refactoring*). Integrirani program za pronalaženje pogrešaka (engl. *debugger*) ispravlja pogreške u izvornom kodu ali i u strojnom kodu. Ostali ugrađeni alati su: dizajner obrazaca za grafičke aplikacije, web dizajner, dizajner klasa i dizajner sheme baze podataka. Visual Studio prihvaća dodatke koji (engl. *plugins*) koji poboljšavaju funkcionalnost na gotovo svim razina, uključujući dodavanje podrške za sustave upravljanja kodom i dodavanje novih alata kao što su uređivači za specifične jezike ili alata za druge aspekte životnog ciklusa razvoja programa.

4.1.4. Razvojni okvir ASP. NET Web API

ASP.NET je razvojni okvir otvorenog koda dizajniran za izradu dinamičkih web stranica, web aplikacija i web usluga. Prva verzija je izdana 2002. godine, a naslijedio je Microsoftovu tehnologiju Active Server Pages [27]. Izgrađen je na CLR sustavu koji dopušta programerima uporabu razvojnog okvira ASP.NET, pišući bilo kojeg jezik kojeg podržava platforma .NET.

ASP.NET Web API je razvojni okvir koji olakšava izgradnju HTTP usluga koje dopiru do širokog raspona klijenata, uključujući preglednike i mobilne uređaje. ASP.NET Web API je idealna platforma za razvoj RESTful aplikacija na platformi .NET [27]. Oslanja se na web standarde, kao što su HTTP, JSON i XML i standardni skup pravila kako bi pružio jednostavan način za izgradnju i izlaganje REST podatkovnih usluga. Koristi kontrolere kao objekte koji obrađuju HTTP zahtjeve. Promatrano iz perspektive arhitekture, ASP.NET Web API vrlo je sličan kosturu ASP.NET MVC jer se oslanja na iste temeljne koncepte, kao što su usmjeravanje, kontroleri pa čak i rezultati akcija kontrolera [28]. On te koncepte, međutim, koristi kako bi podržavao bitno drugačiji skup situacija - situacije koje podrazumijevaju rad s podacima a ne samo generiranje HTML koda.

4.1.5. Baza podataka MongoDB

MongoDB je besplatan program za rad s bazama podataka, otvorenog koda, orijentiran na dokumente i platformski neovisan [29]. MongoDB koristi fleksibilne dokumente slične JSON shemama, što znači da se polja mogu razlikovati od dokumenta do dokumenta, a podatkovna struktura se može mijenjati tijekom vremena. Model dokumenta mapira se s objektima u kodu aplikacije koja radi s MongoDB bazom podatka, što olakšava rad s podacima. MongoDB upiti, indeksiranje i agregacija u stvarnom vremenu pružaju snažne načine pristupa i analize podataka. MongoDB je raspodijeljena baza podataka u svojoj jezgri, pa su visoka dostupnost, horizontalno skaliranje i geografska raspodjela ugrađeni i jednostavni za upotrebu.

Filozofija dizajna MongoDB programa usmjerena je na kombiniranje kritičnih mogućnosti relacijskih baza podataka s inovacijama NoSQL tehnologije [29]. MongoDB pohranjuje podatke u binarnom prikazu kroz tip podatka pod nazivom BSON (engl. *Binary JSON*). BSON proširuje popularnu JSON reprezentaciju tako da uključuje dodatne tipove podataka, kao što su *int, long, date, floating point, decimal128*. MongoDB ima tendenciju da sadrži sve podatke za određeni zapis u jednom dokumentu, dok se u relacijskim bazama podataka informacija za određeni zapis obično širi preko mnogo tablica. No, također omogućuje razvojnim programerima dizajniranje i razvoj sheme dokumenta kroz iterativan i okretan pristup, istodobno provodeći upravljanje podacima. Za situacije u kojima su potrebna stroga jamstva o strukturi podataka i sadržaju dokumenta, MongoDB omogućuje validaciju shema.

4.1.6. Okvir za ubrizgavanje ovisnosti Ninject

Ninject je brz, veoma lagan, okvir za ubrizgavanje ovisnosti u .NET aplikacije [30]. Pomaže podijeliti aplikaciju u zbirku labavo povezanih, visoko kohezivnih komponenata, a zatim ih sastaviti na fleksibilan način. Uporabom okvira Ninject za podršku arhitekturi programa, programski kod postaje lakši za pisanje, ponovnu upotrebu, testiranje i izmjenu. Umjesto ručnog pisanja koda za spajanje komponenata ili proučavanja XML konfiguracijskih datoteka, Ninject omogućuje jednostavno sučelje kompatibilno s prevoditeljem i razvojnim okruženjem. Ninject uvodi izuzetno moćan i fleksibilan oblik vezivanja (engl. *bindings*) nazvan kontekstualnim vezivanjem. Umjesto vezivanja na temelju identifikatora niza, Ninject je svjestan okoline aplikacije i može promijeniti implementaciju za određenu uslugu tijekom izvršavanja.

Ninject koristi prevoditelj izraza na platformi .NET i čeka na izraz koji predstavlja zahtjev za nekom uslugom. Kada je zatražen za instancu neke usluge, Ninject pretražuje dostupne implementacije te usluge i izabire onu koja najbolje odgovara svojim parametrima. Ono što se očekuje od razvojnog programera koji koristi Ninject, je da definira vezivanja za sve tipove usluga koje se mogu koristiti u aplikaciji. Vezivanje tipa usluge je mapiranje između tipa usluge i tipa implementacije te usluge. Ubrizgavanje ovisnosti u okviru Ninject se provodi tako da, kad god Ninject bude zatražen neku uslugu, on provjerava vezivanja tipa te usluge i ubrizgava pripadajuću implementaciju usluge.

4.1.7. Okvir za ovjeru autentičnosti ASP.NET Identity

ASP.NET Identity je potpuno prerađen okvir koji dovodi prijašnji okvir za ovjeru autentičnosti, .ASP.NET Membership, u moderno doba [31]. ASP.NET Identity olakšava integraciju različitih sustava za provjeru autentičnosti kao što su lokalno korisničko ime, lozinka i društvene prijave poput Facebook-a, Twitter-a i druge. Također omogućuje veću kontrolu nad postojanim podacima izabranoj pozadinskoj tehnologiji. ASP.NET Identity uvodi modernije sustave provjere autentičnosti kao što je dvostruka faktorska ovjera autentičnosti (engl. *two - factor authentication*). ASP.NET Identity se može koristiti za osiguravanje web, mobilnih i hibridnih aplikacija i web API-ja.

ASP.NET Identity se temelji na OWIN (engl. *Open Web Interface for .NET*) srednjem sloju koji se može koristiti na bilo kojem OWIN baziranom poslužitelju te nema nikakvu ovisnost o System.Web biblioteci. U aplikacijama zasnovanim na okviru ASP.NET Identity, moguća je potpuna kontrola nad shemom podataka korisnika i profila. Prema standardnim postavkama, okvir ASP.NET Identity pohranjuje sve korisničke podatke u bazu podataka, a koristi Entity Framework Code First princip kako bi implementirao mehanizme postojanosti. Također, postoji pružatelj uloga koji omogućuje ograničavanje pristupa dijelovima aplikacije po ulogama. Mogu se jednostavno izraditi uloge poput administratora ili običnog korisnika i pridodati korisnike u uloge. ASP.NET Identity se preraspodjeljuje kao NuGet paket, a može se odabrati pri kreiranju predložaka aplikacija u Visual Studio razvojnom okruženju.

4.1.8. Okvir za objektno-relacijsko mapiranje Entity Framework

Entity Framework je okvir za objektno-relacijsko mapiranje, otvorenog koda, za aplikacije na platformi .NET. Prethodno je bio dio platforme .NET, no šesta verzija je odvojena samostalno [32]. Omogućuje razvojnim programerima rad s podacima pomoću objekata domene specifičnih za određene domene bez fokusiranja na tablice baze podataka i stupce u kojima se ti podaci pohranjuju. Pomoću Entity Framework okvira razvojni programeri mogu raditi na višoj razini apstrakcije kada se bave podacima te mogu stvoriti i održavati aplikacije usmjerene na podatke s manje koda u usporedbi s tradicionalnim aplikacijama.

Značajke Entity Framework okvira su [32]:

- Modeliranje - Entity Framework stvara entitetski model podataka na temelju entiteta POCO (engl. *Plain Old CRL Object*) entiteta sa svojstvima za dohvaćanje i postavljanje.

- Upiti - Entity Framework omogućuje korištenje LINQ upitima za preuzimanje podataka iz baze podataka, a prevodi te upite na jezik upita specifičan za bazu podataka.
- Promjena praćenja - Entity Framework prati promjene koje su se dogodile u instancama entiteta, koje treba podnijeti u bazi podataka.
- Spremanje - Entity Framework izvršava naredbe ubacivanja, ažuriranja i brisanja u bazi podataka na temelji promjena koje su se dogodile na entitetima u aplikaciji.
- Konkurentnost - Entity Framework koristi optimističku konkurenciju prema zadanim postavkama kako bi zaštitio nadopune promjene koje je izvršio drugi korisnik, budući da su podaci preuzeti iz baze podataka.
- Transakcije - Entity Framework obavlja automatsko upravljanje transakcijama prilikom upita ili spremanja podataka.
- Konfiguracije - Entity Framework omogućuje konfiguraciju modela pomoću atributa ili API-ja za platformu .NET, za nadjačavanje zadanih konvencija.
- Migracije - Entity Framework pruža skup naredbi za migraciju, kako bi stvorili ili upravljali shemom baze podataka.

4.1.9. Baza podataka SQL Server

SQL Server je sustav za upravljanje relacijskim bazama podataka koji je razvio Microsoft, pa se još naziva i Microsoft SQL Server (MSSQL) [34]. Riječ je o programskom proizvodu koji ima primarnu funkciju pohranjivanja i dohvaćanja podataka kao što to zahtijevaju druge programske aplikacije. Microsoft prodaje desetak različitih izdanja sustava MSSQL, namijenjenih različitoj publici i za različita opterećenja. Baza podataka SQL Server je prvenstveno dizajnirana za natjecanje s konkurentima Oracle Database i MySQL.

Kao i svi sustavi za upravljanje relacijskim bazama podataka, SQL Server podržava standardni SQL jezik ANSI SQL. Također, SQL Server sadrži i vlastitu implementaciju SQL jezika, T-SQL. Primjeri nekih značajki koje uključuje SQL Server su: podrška za format podataka XML, prikazi dinamičkog upravljanja (engl. *dynamic management views*), sposobnost pretraživanja cijelog teksta i zrcaljenje baza podataka. SQL Server je prvenstveno izgrađen oko tablične strukture koja povezuje elemente podataka u različite tablice, izbjegavajući potrebu redundantnog pohranjivanja podataka. Relacijski model također pruža referencijalni integritet i ostala ograničenja integriteta za održavanje točnosti podataka. SQL Server se pridržava načela atomnosti, konzistentnosti,

izolacije i izdržljivosti - zajednički poznate kao svojstva ACID, a osmišljeni su tako da jamče pouzdanost obrađivanja transakcija baze podataka.

Microsoft SQL Server je bio dostupan isključivo na operacijskom sustavu Windows više od 20 godina [34]. No, 2016. godine Microsoft je istaknuo kako planira napraviti sustav za upravljanje relacijskim bazama podataka dostupan i na operacijskom sustavu Linux. Kasnije je to ažuriranje formalno nazvano SQL Server 2017, a postalo je opće dostupno u listopadu te godine.

4.1.10. Rješenje za pohranu objekata Azure Blob Storage

Azure Blob Storage je Microsoftovo rješenje za pohranu objekata u oblaku. Optimizirano je za pohranu masivnih količina nestrukturiranih podataka, kao što su tekstualni podaci ili binarni podaci [35]. Glavne značajke koje omogućava Azure Blob Storage su:

- Posluživanje slika ili dokumenata izravno kroz preglednik,
- Pohranjivanje datoteka za raspodijeljeni pristup,
- Prijenos zvukovnih zapisa i videozapisa,
- Pisanje u zapisničku datoteku,
- Pohranjivanje podataka za sigurnosno kopiranje i vraćanje, oporavak od katastrofe i arhiviranje,
- Pohranjivanje podataka za analizu lokalnim ili Azure servisom.

Objektima u pohrani se može pristupiti s bilo koje lokacije preko HTTP ili HTTPS protokola.

Azure Blob Storage izlaže tri vrste resursa: račun za pohranu, spremnike unutar računa i objekte u spremniku. Sav pristup podatkovnim objektima u pohrani se događa putem računa za pohranu. Spremnik organizira skup objekata i sličan je mapi u datotečnom sustavu. Svi objekti se nalaze u spremniku. Račun za pohranu može sadržavati neograničen broj spremnika, a spremnik može pohraniti neograničen broj objekata. Azure Blob Storage raspolaže s tri vrste objekata: blokovi objekata, objekti za nadodavanje i objekti za straničenje. Blokovi objekata pohranjuju tekst i binarne podatke, do 4,7 TB. Objekti za nadodavanje su optimizirani za dodatne operacije, kao što je zapisivanje podataka s virtualnih strojeva. Objekti za straničenje pohranjuju podatke s izravnim pristupom do veličine od 8 TB.

4.2. BaaS usluga pohrane podataka

BaaS usluga pohrane podataka omogućuje klijentu stvaranje i manipuliranje podatkovnim resursima, konzumiranjem RESTful API-ja. To znači da su resursi neovisni o tehnologiji i

platformi i dostupni su preko HTTP zahtjeva bez pisanja prilagođenog pozadinskog koda. Osnovna jedinica podataka je upravo resurs, a resursi su organizirani u zbirke podataka. Svaka zbirka je skup parova ključ - vrijednost koji su pohranjeni u formatu JSON. U ovoj BaaS usluzi, struktura zbirke se definira shemom, koja je uz resurs druga jedinica podataka usluge. Shema sadrži metapodatke o zbirci, te validacijska pravila o njenoj strukturi. BaaS usluga pohrane podataka je visoko skalabilan sustav za pohranu podataka, a temelji se na NoSQL MongoDB bazi podataka. Upotrebom te tehnologije, ostavlja se korisniku prostor za definiranjem strukture baze njegove aplikacije koristeći samo format JSON i osnovne operacije BaaS usluge. Usluga pohrane podataka se dijeli na dvije funkcionalnosti:

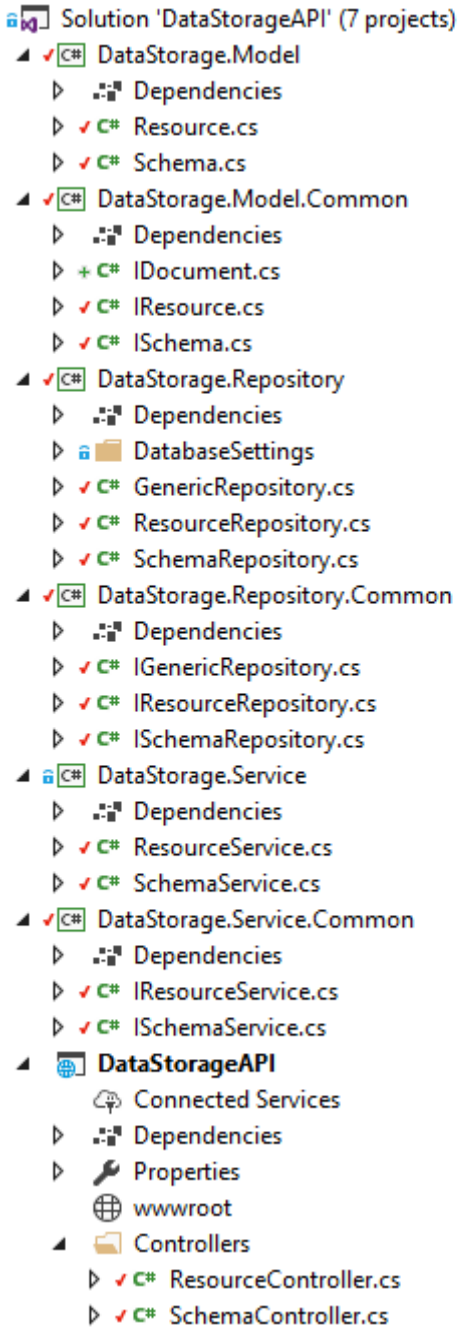
- Rad s podatkovnim resursima usluge
- Rad sa shemama zbirki podataka

4.2.1. Struktura usluge pohrane podataka

BaaS usluga pohrane podataka implementirana je kroz razvojni okvir ASP.NET Web API. Usluga je dizajnirana na temelju višeslojne arhitekture *Onion*, na način da je usluga podijeljena na četiri sloja. Slika 4.1 prikazuje sadržaj programskog rješenja BaaS usluge pohrane podataka u obliku skupa projekata razvijenih na platformi .NET. Aplikacijski sloj usluge je web API, a preostala tri sloja su: sloj usluga, sloj repozitorija i sloj modela. Potonja tri sloja su prema vrsti projekta biblioteke klasa. Za svaki od potonja tri sloja, postoji jedan projekt, također po vrsti biblioteka klasa, kojem se u nazivu dodaje riječ zajednički (engl. *common*), a služi kao reprezentacija pripadajućeg sloja nižim i višim slojevima. U zajedničkom projektu nekog sloja, nalaze se definirana sučelja tog sloja, preko kojih gornji slojevi komuniciraju s tim slojem. To znači da gornji slojevi koriste samo sučelja donjih slojeva, a tijekom izvođenja se sučelja zamjenjuju konkretnim implementacijama tih slojeva, koristeći mehanizme ubrizgavanja ovisnosti na razvojnom okviru ASP.NET.

Za ostvarivanje mehanizma ubrizgavanja ovisnosti koristi se okvir Ninject. Unutar razvojnog okvira ASP.NET Web API, Ninject se instalira u obliku NuGet paketa. NuGet je besplatan upravitelj paketa, otvorenog koda, dizajniran za Microsoftovu razvojnu platformu. Da bi se ostvarilo načelo ubrizgavanja ovisnosti, potrebno je registrirati sva vezivanja sučelja i implementacije, prije pokretanja aplikacije. Ninject ima ugrađene atribute koji omogućuju biranje funkcija koje se izvode na samom početku i kraju aplikacije. U funkciji koja se izvodi na početku aplikacije, inicijalizira se objekt klase *Bootstrapper*, koji služi za registriranje svih modula u kojim

se definiraju vezivanja sučelja i implementacija. Primjer jednog takvog modula za sloj repozitorija, prikazan je na slici 4.2.



Sl. 4.1. Sadržaj programskog rješenja BaaS usluge pohrane podataka

U BaaS sustavu pohrane podataka, koristi se ubrizgavanje na razini konstruktora kao metoda ubrizgavanja ovisnosti. Na slici 4.3 prikazan je programski kod u kojem se definira konstruktor za kontroler naziva *ResourceController*. Kontroler koristi funkcije definirane sučeljem *IResourceService*, koje se nalazi u sloju usluga. Tijekom instanciranja kontrolera, sučelje se zamjenjuje konkretnom implementacijom i to onom koja je definirana u vezivanju za to sučelje.

Sva vezivanja nekog sloja se nalaze u klasi naziva *DIModule*, a zatim se klase kao moduli registriraju pomoću okvira za ubrizgavanje ovisnosti Ninject, na samom početku izvođenja aplikacije.

```
using Ninject.Extensions.Factory;
using Project.Repository.Common;

namespace Project.Repository
{
    public class DIModule : Ninject.Modules.NinjectModule
    {
        public override void Load()
        {
            Bind<IGenericRepository>().To<GenericRepository>();
            Bind<IResourceRepository>().To<ResourceRepository>();
            Bind<ISchemaRepository>().To<SchemaRepository>();
        }
    }
}
```

Sl. 4.2. Modul ubrizgavanja ovisnosti

```
1 reference | MarioDudjak, 36 days ago | 1 author, 2 changes
public class ResourceController : Controller
{

    #region Properties

    6 references | MarioDudjak, 36 days ago | 1 author, 1 change | 0 exceptions
    protected IResourceService ResourceService { get; private set; }

    #endregion Properties

    #region Constructors

    0 references | MarioDudjak, 36 days ago | 1 author, 1 change | 0 exceptions
    public ResourceController(IResourceService resourceService)
    {
        ResourceService = resourceService;
    }

    #endregion Constructors
```

Sl. 4.3. Primjena metode ubrizgavanja ovisnosti na razini konstruktora

Aplikacijski sloj BaaS usluge pohrane podataka, u obliku web API-ja, sadrži dva kontrolera: *ResourceController*, koji radi s resursima i *SchemaController*, koji radi sa shemama. Oba kontrolera sadrže asinkrone funkcije za izvođenje četiriju osnovnih HTTP operacija te još jednu metodu za pretraživanje podataka na osnovu zadanog kriterija. Sve funkcije navedenih kontrolera vraćaju odgovor u formatu JSON s pripadnim HTTP statusnim kodovima. Funkcije primaju podatke iz putanje, tijela i zaglavlja zahtjeva.

4.2.2. Dizajn baze podataka usluge pohrane podataka

Za bazu podataka usluge pohrane podataka izabrana je baza podataka MongoDB. Dizajn baze podataka je proveden, imajući u vidu da sustav BaaS mogu istovremeno koristiti više aplikacija. Jedno od općih i širokih pitanja, koje se pojavljuje prilikom dizajna baze podataka MongoDB je razlaganje baze podataka na više manjih baza ili na više zbirki. U ovom radu, izabrano je razlaganje na više zbirki zbog manje složenosti održavanja baza podataka i njihovih poslužitelja. Drugo pitanje u području dizajna baze podataka MongoDB, tiče se performansi baze podataka. Prema [28], MongoDB je više predviđena za horizontalnu skalabilnost u odnosu na vertikalnu skalabilnost. To znači da je, u smislu performansi, poželjno koristiti jednu zbirku podataka i sve dokumente spremati u tu zbirku. No, zbog poslovnih zahtjeva usluge pohrane podataka, nemoguće je koristiti jednu zbirku, za pohranu podataka koji pristižu iz različitih aplikacija. Klijent sustava BaaS mora imati mogućnost definiranja sheme zbirke, tj. validacijskih pravila koji se primjenjuju na dokumente unutar zbirke. No, ne smije se dozvoliti nametanje tuđih pravila drugom klijentu, stoga se podaci ne mogu spremati u jednu zbirku. Zato je baza podataka dizajnirana tako da svaki klijent ima mogućnosti kreiranja vlastitih zbirki u koje može spremati podatke po vlastitom izboru.

S druge strane, upotrebom nerelacijske baze podataka, kao što je MongoDB, gube se funkcionalnosti definiranja struktura tablica i odnosa među njima, što je omogućeno u relacijskim bazama podataka. No, ako bi se u ovoj usluzi koristila relacijska baza podataka, klijentu bi se morala omogućit funkcionalnost definiranja struktura tablica i odnosa među njima. Time bi se ova usluga više približila konceptu nekog razvojnog okvira, pisanom u određenom pozadinskom programskom jeziku. S obzirom da je jedna od glavnih svrha sustava BaaS olakšan razvoj programskih rješenja, predstavljanje sustava BaaS kao razvojnog okvira ne ispunjava tu svrhu. Stoga je izabrana nerelacijska baza podataka i time je naglašena fleksibilnost klijenta prilikom rada s podacima u bazi podataka, a istovremeno je olakšan razvoj i ispunjena jedna od glavnih svrha sustava BaaS.

U razvojnom okviru ASP.NET Web API, s bazom podataka MongoDB se upravlja koristeći upravljački program (engl. *driver*) MongoDB C#. Unutar tog upravljačkog programa, baza podataka se predstavlja sučeljem tipa *IDatabase,* a dokumenti u bazi podataka se mapiraju u tip podatka *BsonDocument*. Web API koristi domenske modele pisane u programskom jeziku C# za predstavljanje logike podataka i za lakši rad s podacima unutar razvojnog okvira. Razvijena su dva domenska modela: Resurs i Shema. Resurs je ekvivalent dokumentu unutar baze podataka MongoDB, a sadrži sljedeća svojstva:

- Jedinstveni identifikator (*_id*),
- Podatkovni objekt predstavljen kao JSON objekt (*Data)*,
- Vrijeme kreiranja (*DateCreated*),
- Vrijeme zadnje izmjene (*DateModified*).

Shema je ekvivalentna zbirci unutar baze podataka MongoDB, a sadrži sljedeća svojstva:

- Jedinstveni identifikator (*_id*),
- Naziv sheme (*Name*),
- Opis sheme (*Description*),
- Validacijski objekt predstavljen kao JSON objekt (*Validator*),
- Razina validacije (*ValidatorLevel*),
- Vrijeme kreiranja (*DateCreated*),
- Vrijeme zadnje izmjene (*DateModified*).

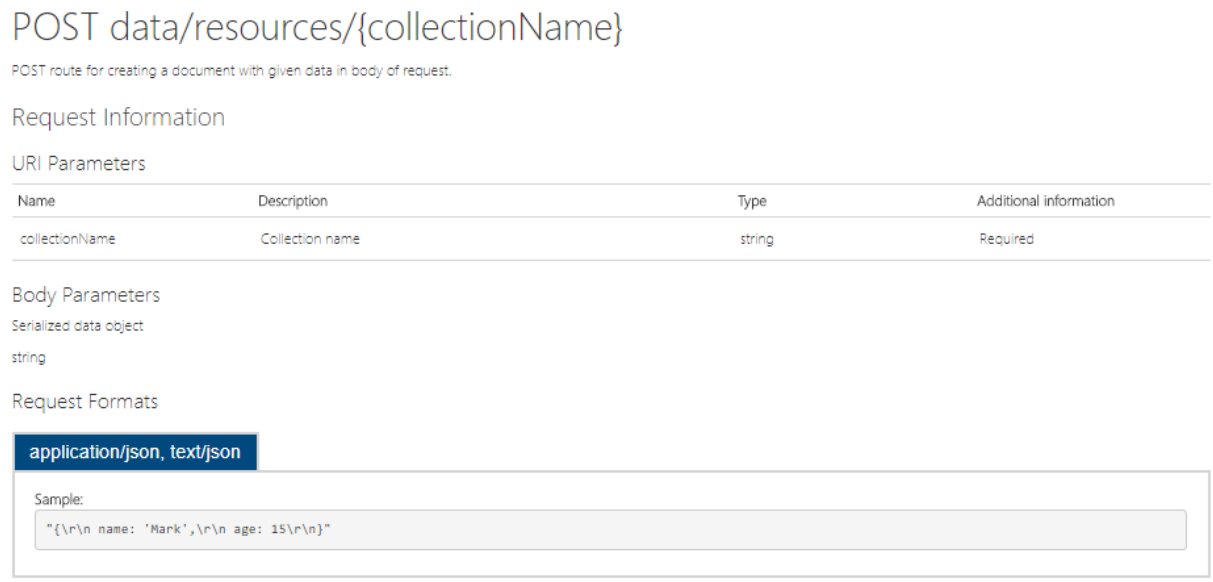
4.2.3. Rad s podatkovnim resursima usluge

Osnovna jedinica podataka unutar usluge pohrane podataka je resurs. Resurs je unutar web API-ja predstavljen domenskim modelom koristeći tipove podataka iz programskog jezika C#. Prilikom komunikacije s bazom podataka, izvodi se mapiranje modela resursa, u poseban tip podatka naziva *BsonDocument*, koji predstavlja dokument iz baze podataka u programskom jeziku C#. Usluga pohrane podataka kroz okvir web API pruža metode za rad s podatkovnim resursima usluge. Cjelokupni set CRUD (engl. *Create, Read, Update and Delete*) operacija dostupan je za manipuliranje resursima. Svaka operacija se dohvaća putem REST zahtjeva, pri čemu svaki zahtjev mora sadržavati sljedeće parametre:

- Naziv HTTP operacije,
- Krajnju točku ili URL operacije,
- Polje *Authorization* u zaglavlju s vrijednosti značke za autorizaciju,
- Polje *Content-Type* u zaglavlju postavljeno na *application/json*.

Uz navedene obvezne podatke, neke operacije mogu zahtijevati i dodatne parametre.

Kreiranje resursa je prva operacija u radu s podatkovnim resursima. Na slici 4.4 prikazana je definicija zahtjeva kojim se izvodi operacija kreiranja resursa. Za kreiranje resursa, potrebni parametri su ime zbirke i podatkovni objekt. Ime zbirke se dohvaća iz URL-a zahtjeva, a podatkovni objekt iz tijela zahtjeva. Podatkovni objekt je serijalizirani JSON objekt iz razloga lakšeg manipuliranja resursom unutar usluge.

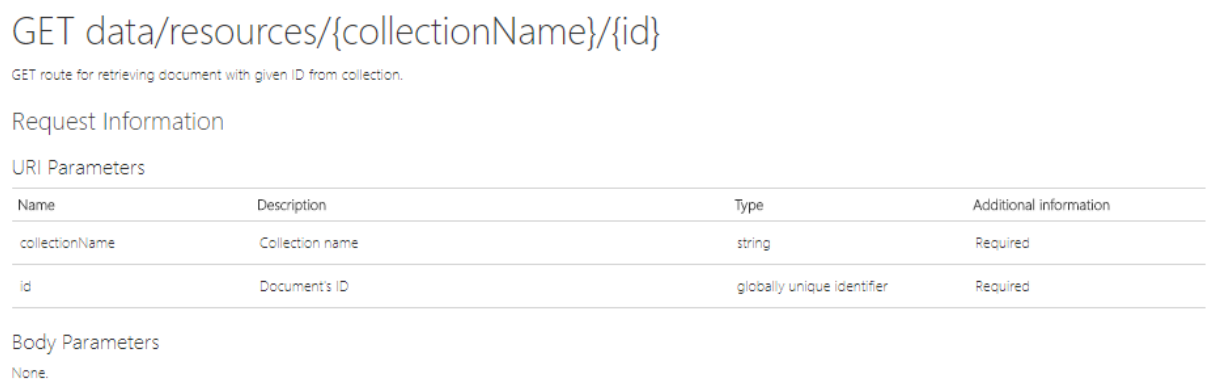


Sl. 4.4. Definicija REST zahtjeva za kreiranje podatkovnog resursa

Rezultat izvođenja operacije kreiranja resursa je REST odgovor, čija je definicija prikazana na slici 4.5. Kreirani resurs je smješten u tijelo REST odgovora, a predstavljen kao JSON objekt. Sadrži novostvorene podatke: identifikator resursa, vrijeme kreiranja i vrijeme zadnje izmjene.

Sl. 4.5. Definicija REST odgovora nakon kreiranja podatkovnog resursa

Dohvaćanje resursa je druga operacija u radu s podatkovnim resursima. Na slici 4.6 prikazana je definicija zahtjeva kojim se izvodi operacija dohvaćanja resursa. Za dohvaćanje resursa, potrebni parametri su ime zbirke i identifikator resursa, koji se dohvaćaju iz URL-a zahtjeva.



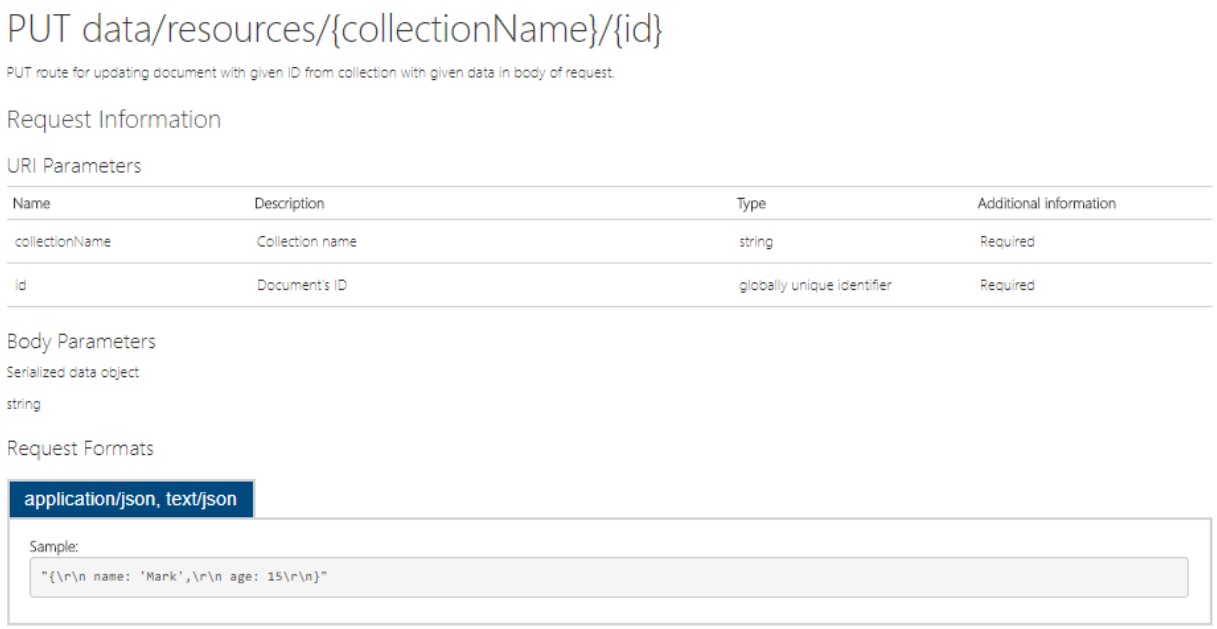
Sl. 4.6. Definicija REST zahtjeva za dohvaćanjem podatkovnog resursa

Rezultat izvođenja operacije dohvaćanja resursa je REST odgovor, čija je definicija prikazana na slici 4.7. Traženi resurs je smješten u tijelo REST odgovora, a predstavljen kao JSON objekt. Svojstva resursa su identifikator, podatkovni objekt, vrijeme kreiranja i posljednje vrijeme izmjene resursa. Podatkovni objekt je serijaliziran u znakovni niz radi lakšeg rada s resursom unutar usluge.

Sl. 4.7. Definicija REST odgovora nakon dohvaćanja podatkovnog resursa

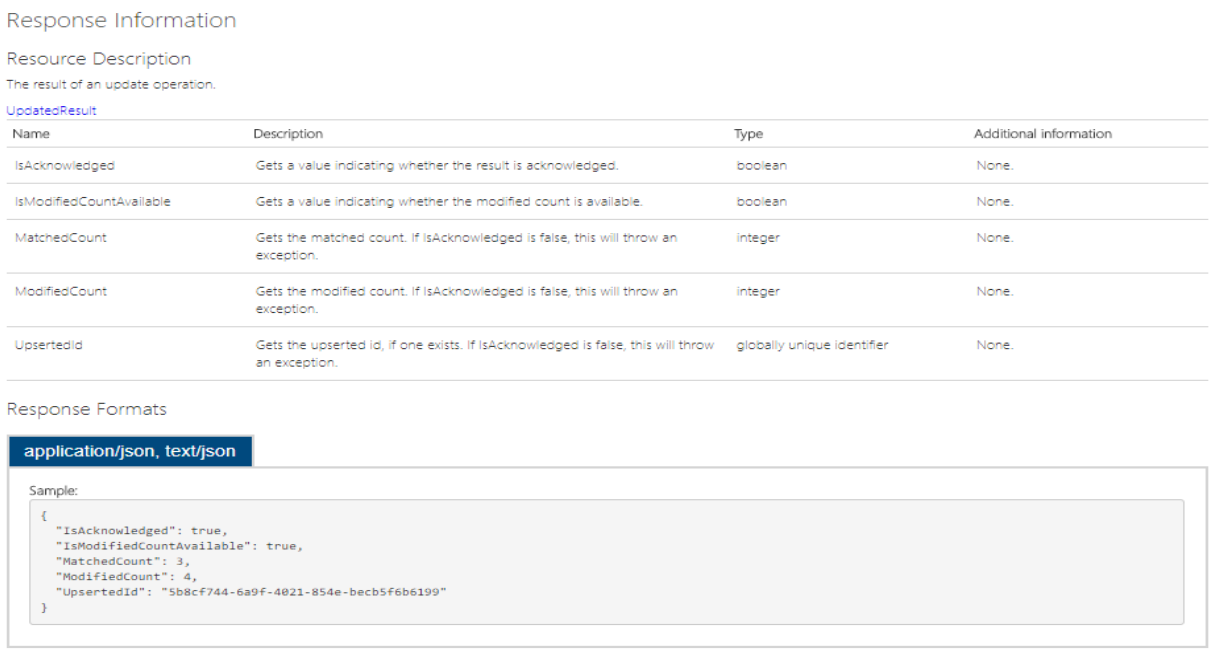
Ažuriranje resursa je treća operacija u radu s podatkovnim resursima. Na slici 4.8 prikazana je definicija zahtjeva kojim se izvodi operacija ažuriranja resursa. Za ažuriranje resursa, potrebni

parametri su ime zbirke, identifikator resursa i podatkovni objekt. Ime zbirke i identifikator resursa se dohvaćaju iz URL-a zahtjeva, a podatkovni objekt se dohvaća iz tijela zahtjeva.



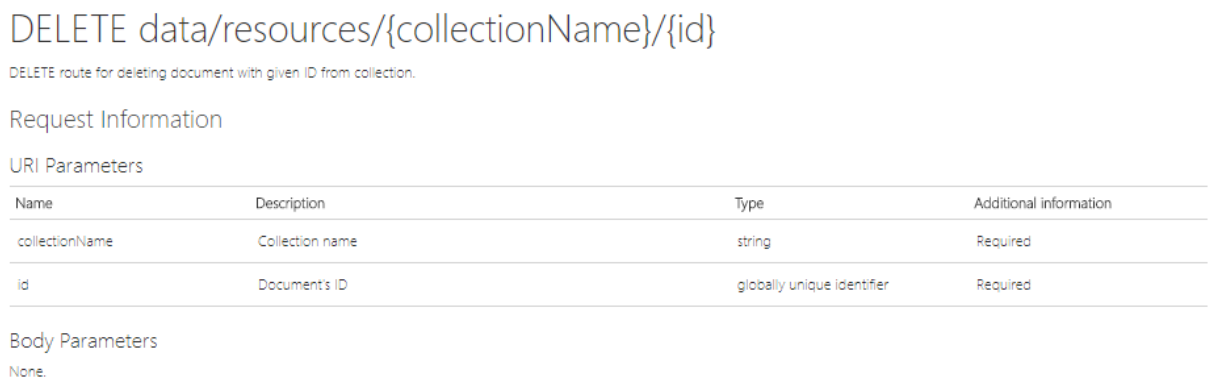
Sl. 4.8. Definicija REST zahtjeva za ažuriranjem podatkovnog resursa

Rezultat izvođenja operacije ažuriranja resursa je REST odgovor, čija je definicija prikazana na slici 4.9. U tijelo odgovora smješten je JSON objekt koji prikazuje rezultat operacije ažuriranja. Svojstva objekta su identifikator ažuriranog resursa, vrijednost koja indicira uspješnog operacije, vrijednost koja indicira je li dostupan broj ažuriranih resursa, broj pronađenih resursa i broj ažuriranih resursa.



Sl. 4.9. Definicija REST odgovora nakon ažuriranja podatkovnog resursa

Brisanje resursa je četvrta operacija u radu s podatkovnim resursima. Na slici 4.10 prikazana je definicija zahtjeva kojim se izvodi operacija brisanja resursa. Za brisanje resursa, potrebni parametri su ime zbirke i identifikator resursa, koji se dohvaćaju iz URL-a zahtjeva.



DELETE data/resources/{collectionName}/{id}

DELETE route for deleting document with given ID from collection.

Request Information

URI Parameters

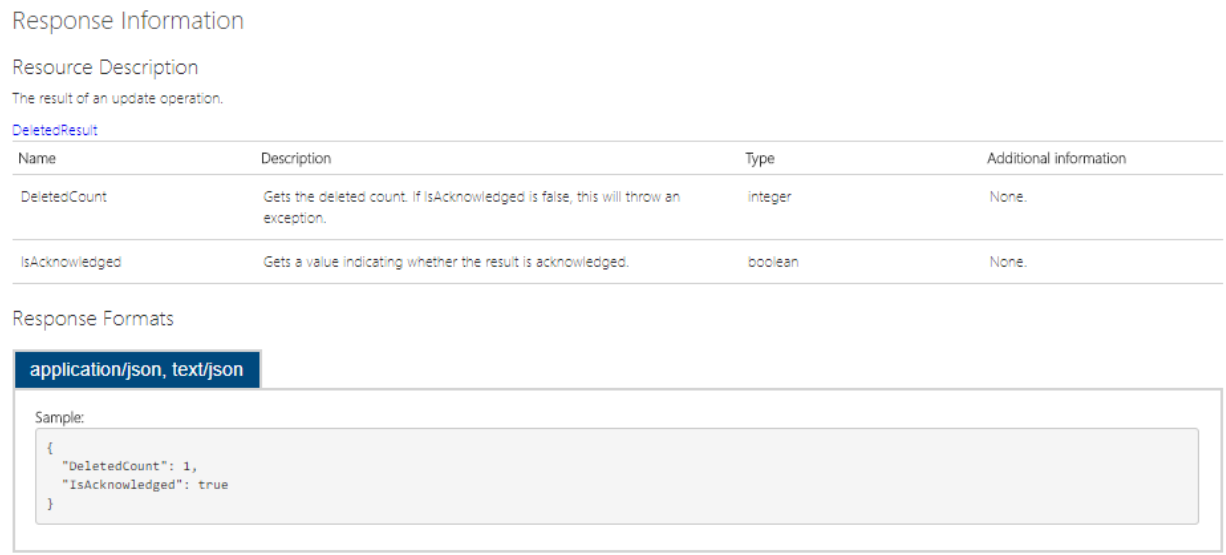
Name	Description	Type	Additional information
collectionName	Collection name	string	Required
id	Document's ID	globally unique identifier	Required

Body Parameters

None.

Sl. 4.10. Definicija REST zahtjeva za brisanjem podatkovnog resursa

Rezultat izvođenja operacije brisanja resursa je REST odgovor, čija je definicija prikazana na slici 4.11. U tijelo odgovora smješten je JSON objekt koji prikazuje rezultat operacije brisanja. Svojstva objekta su broj obrisanih resursa i vrijednost koja indicira uspješnost operacije brisanja.



Response Information

Resource Description

The result of an update operation.

DeletedResult

Name	Description	Type	Additional information
DeletedCount	Gets the deleted count. If IsAcknowledged is false, this will throw an exception.	integer	None.
IsAcknowledged	Gets a value indicating whether the result is acknowledged.	boolean	None.

Response Formats

application/json, text/json

Sample:

```
{
  "DeletedCount": 1,
  "IsAcknowledged": true
}
```

Sl. 4.11. Definicija REST odgovora nakon brisanja podatkovnog resursa

Dohvaćanje resursa prema filtru je peta operacija u radu s podatkovnim resursima. Na slici 4.12 prikazana je definicija zahtjeva kojim se izvodi operacija dohvaćanja resursa prema filtru. Za dohvaćanje resursa prema filtru, potrebni parametri su ime zbirke, upit za pretraživanje, broj stranica s rezultatima, broj rezultata po stranici i vrijednost koja indicira oblik sortiranja. Svi parametri se dohvaćaju iz URL-a zahtjeva.

GET data/resources/{collectionName}?searchQuery={searchQuery}&page={page}&rpp={rpp}&sort={sort}

GET route for retrieving document that matches given query filter.

Request Information

URI Parameters

Name	Description	Type	Additional information
collectionName	Collection name	string	Required
searchQuery	Search query value	string	Required
page	Number of result page.	integer	Required
rpp	Number of results per page.	integer	Required
sort	Value that indicates sorting order	string	Required

Body Parameters

None.

Sl. 4.12. Definicija REST zahtjeva za dohvaćanjem podatkovnog resursa prema filtru

Rezultat izvođenja operacije dohvaćanja resursa prema filtru je REST odgovor, čija je definicija prikazana na slici 4.13. Resursi koji odgovaraju filtru su smješteni u tijelo REST odgovora, a predstavljeni kao niz JSON objekata.

Response Information

Resource Description

The list of documents that matches given filter.

Collection of ResourceVM

Name	Description	Type	Additional information
_id	Unique identifier of document in database.	globally unique identifier	None.
Data	Data field of document in database. Serialized JSON object.	string	None.
DateCreated	Creation date in UTC format.	date	None.
DateModified	Last modification date in UTC format.	date	None.

Response Formats

application/json, text/json

Sample:


```
[
  {
    "_id": "daffdaf2-0d2a-439e-a075-5411f37b2b58",
    "Data": "{\r\n name: 'Mark',\r\n age: 15\r\n}",
    "DateCreated": "2018-06-12T11:55:30.7021336+02:00",
    "DateModified": "2018-06-12T11:55:30.7021336+02:00"
  },
  {
    "_id": "daffdaf2-0d2a-439e-a075-5411f37b2b58",
    "Data": "{\r\n name: 'Mark',\r\n age: 15\r\n}",
    "DateCreated": "2018-06-12T11:55:30.7021336+02:00",
    "DateModified": "2018-06-12T11:55:30.7021336+02:00"
  }
]
```


Sl. 4.13. Definicija REST odgovora nakon dohvaćanja podatkovnog resursa prema filtru

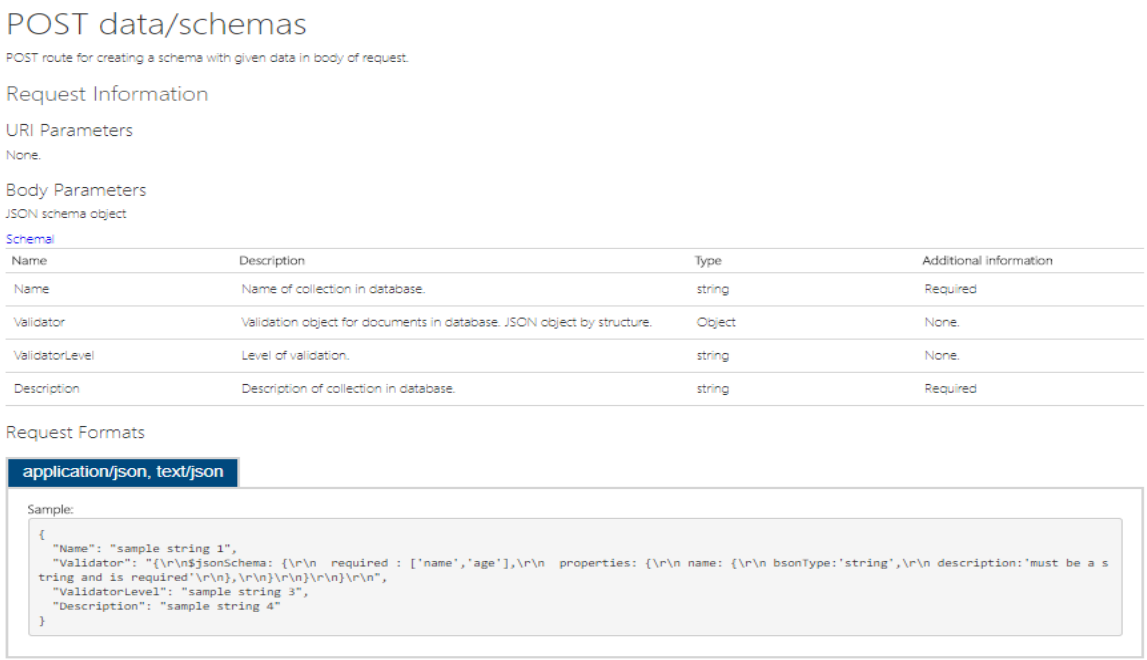
4.2.4. Rad sa shemama zbirki podataka

Druga jedinica podataka, uz resurs, unutar usluge pohrane podataka je shema. Shema definira strukturu zbirke resursa. Unutar web API-ja predstavljena je domenskim modelom koristeći tipove podataka iz programskog jezika C#. Usluga pohrane podataka kroz okvir web API pruža metode za rad sa shemama zbirki podataka. Cjelokupni set CRUD operacija dostupan je za manipuliranje shemama. Svaka operacija se dohvaća putem REST zahtjeva, pri čemu svaki zahtjev mora sadržavati sljedeće parametre:

- Naziv HTTP operacije,
- Krajnju točku ili URL operacije,
- Polje *Authorization* u zaglavlju s vrijednosti značke za autorizaciju,
- Polje *Content-Type* u zaglavlju postavljeno na *application/json*.

Uz navedene obvezne podatke, neke operacije mogu zahtijevati i dodatne parametre.

Kreiranje sheme je prva operacija u radu sa shemama zbirki podataka. Na slici 4.14 prikazana je definicija zahtjeva kojim se izvodi operacija kreiranja sheme. Za kreiranje sheme, potreban parametar je objekt sheme, koji se dohvaća iz tijela zahtjeva. Objekt sheme sadrži parametre: ime sheme, opis sheme, validacijski objekt i razinu validacije. Validacijski objekt je JSON objekt koji ima određenu strukturu, a služi za definiranje validacijskih pravila koji će se primijeniti na nove resurse prilikom ubacivanja u zbirku.



POST data/schemas

POST route for creating a schema with given data in body of request.

Request Information

URI Parameters

None.

Body Parameters

JSON schema object

Schema

Name	Description	Type	Additional information
Name	Name of collection in database.	string	Required
Validator	Validation object for documents in database. JSON object by structure.	Object	None.
ValidatorLevel	Level of validation.	string	None.
Description	Description of collection in database.	string	Required

Request Formats

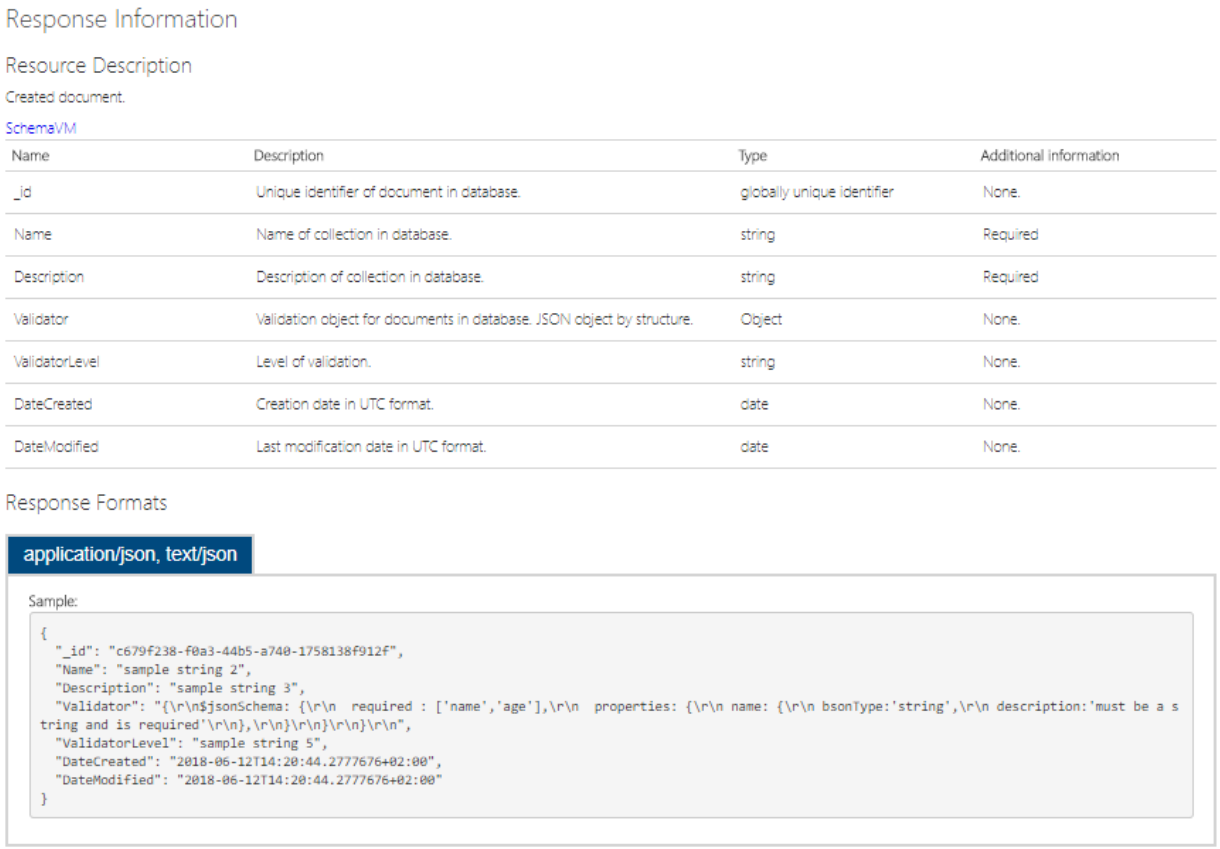
application/json, text/json

Sample:

```
{
  "Name": "sample string 1",
  "Validator": "{\r\n$jsonSchema: {\r\n  required : ['name','age'],\r\n  properties: {\r\n name: {\r\n bsonType:'string',\r\n description:'must be a s
tring and is required'\r\n},\r\n}\r\n}\r\n}\r\n",
  "ValidatorLevel": "sample string 3",
  "Description": "sample string 4"
}
```

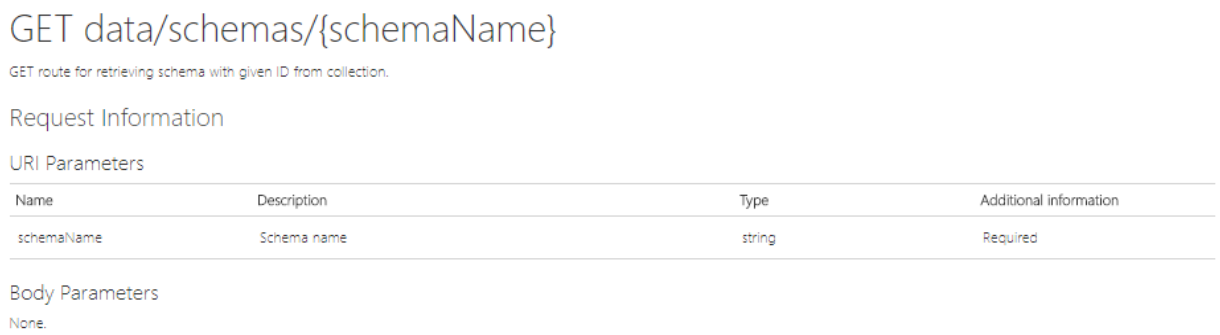
Sl. 4.14. Definicija REST zahtjeva za kreiranje sheme

Rezultat izvođenja operacije kreiranja sheme je REST odgovor, čija je definicija prikazana na slici 4.15. Kreirana shema je smještena u tijelo REST odgovora, a predstavljena kao JSON objekt. Sadrži novostvorene podatke: identifikator sheme, vrijeme kreiranja i vrijeme zadnje izmjene.



Sl. 4.15. Definicija REST odgovora nakon kreiranja sheme

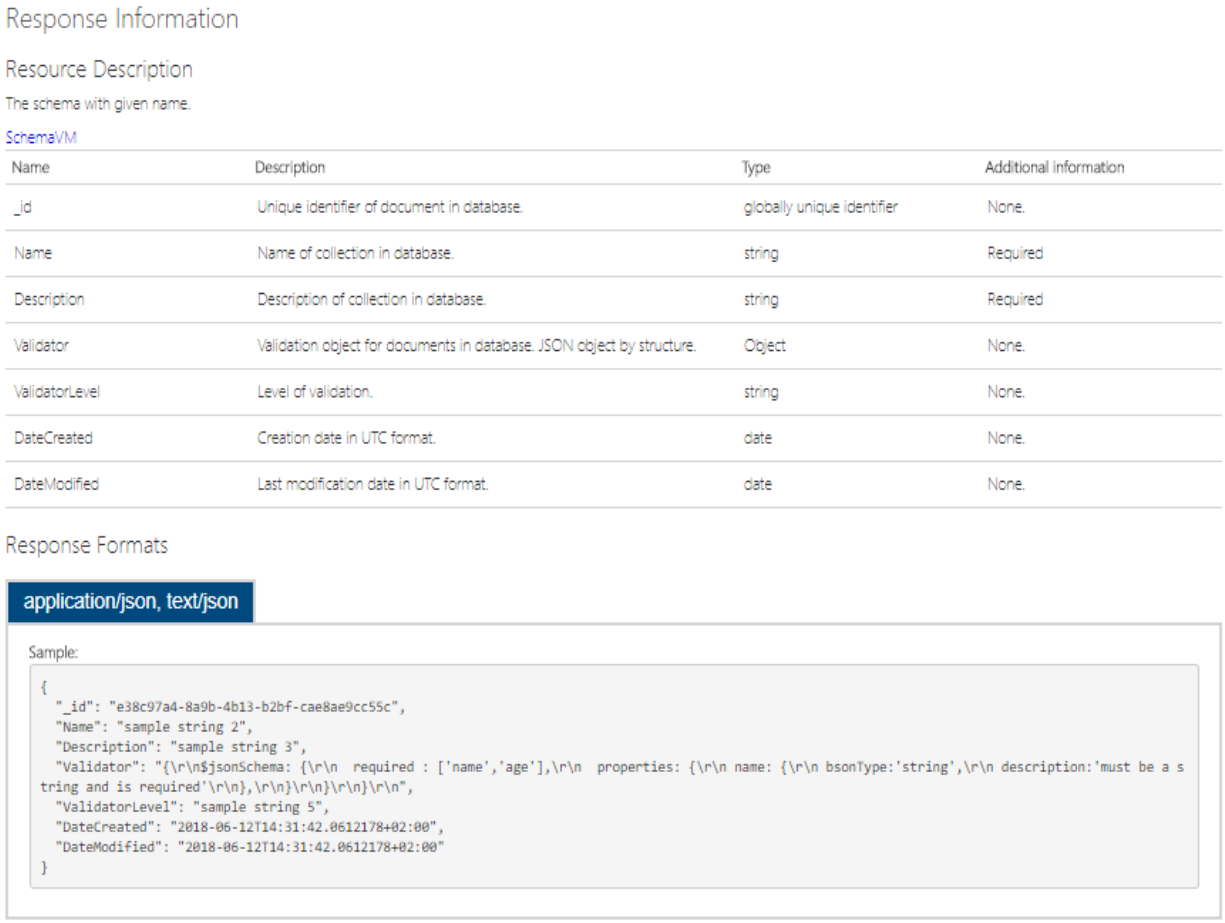
Dohvaćanje sheme je druga operacija u radu sa shemama zbirki podataka. Na slici 4.16 prikazana je definicija zahtjeva kojim se izvodi operacija dohvaćanja sheme. Za dohvaćanje sheme, potreban parametar je naziv sheme, koji se dohvaća iz URL-a zahtjeva.



Sl. 4.16. Definicija REST zahtjeva za dohvaćanje sheme

Rezultat izvođenja operacije dohvaćanja sheme je REST odgovor, čija je definicija prikazana na slici 4.17. Tražena shema je smještena u tijelo REST odgovora, a predstavljena kao JSON objekt.

Svojstva sheme su identifikator, naziv sheme, opis sheme, validacijski objekt, razina validacije, vrijeme kreiranja i posljednje vrijeme izmjene sheme.



Response Information

Resource Description

The schema with given name.

SchemaVM

Name	Description	Type	Additional information
_id	Unique identifier of document in database.	globally unique identifier	None.
Name	Name of collection in database.	string	Required
Description	Description of collection in database.	string	Required
Validator	Validation object for documents in database. JSON object by structure.	Object	None.
ValidatorLevel	Level of validation.	string	None.
DateCreated	Creation date in UTC format.	date	None.
DateModified	Last modification date in UTC format.	date	None.

Response Formats

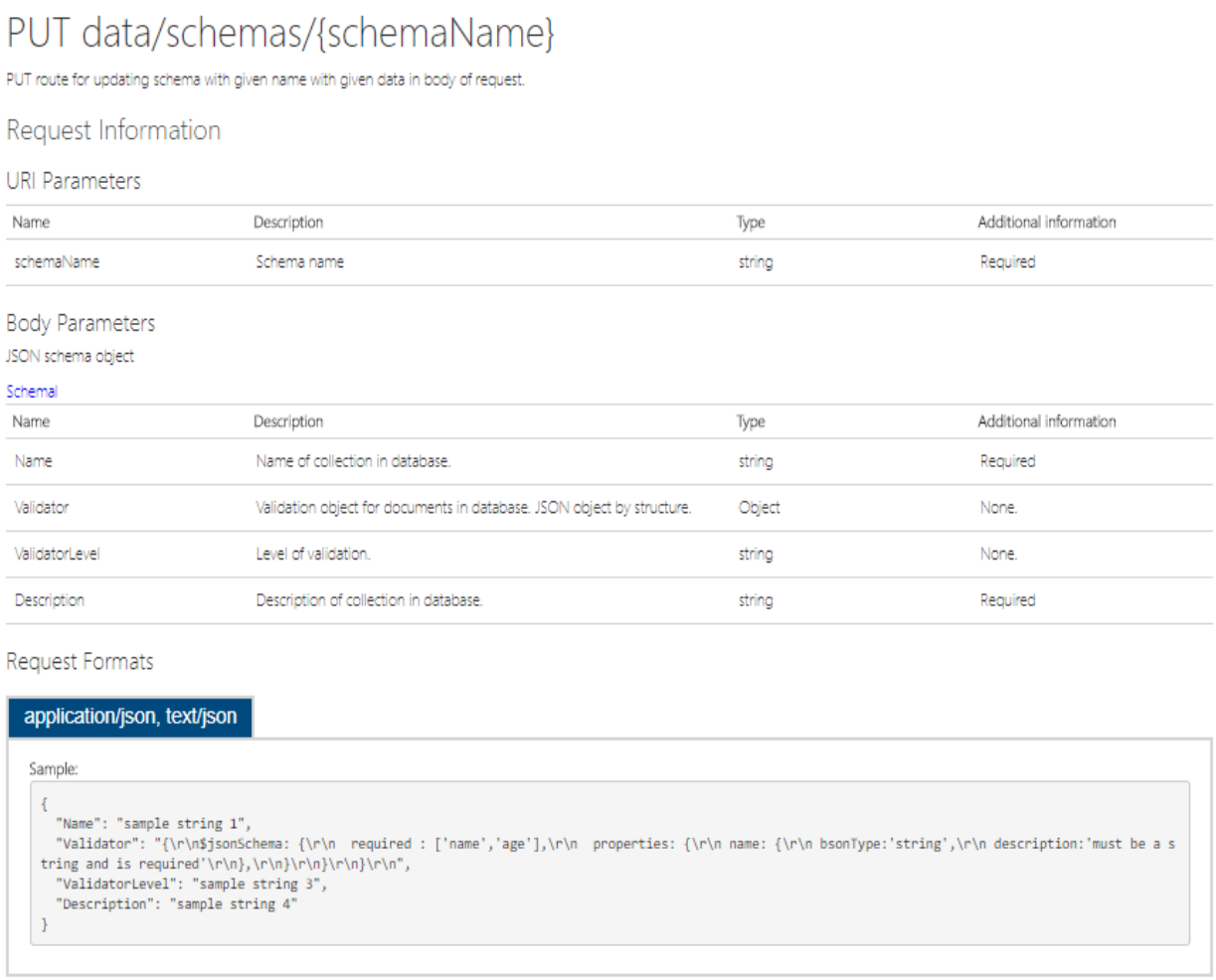
application/json, text/json

Sample:


```
{
  "_id": "e38c97a4-8a9b-4b13-b2bf-cae8ae9cc55c",
  "Name": "sample string 2",
  "Description": "sample string 3",
  "Validator": "{\r\n$jsonSchema: {\r\n  required : ['name','age'],\r\n  properties: {\r\n name: {\r\n bsonType:'string',\r\n description:'must be a s
tring and is required'\r\n},\r\n}\r\n}\r\n}\r\n",
  "ValidatorLevel": "sample string 5",
  "DateCreated": "2018-06-12T14:31:42.0612178+02:00",
  "DateModified": "2018-06-12T14:31:42.0612178+02:00"
}
```

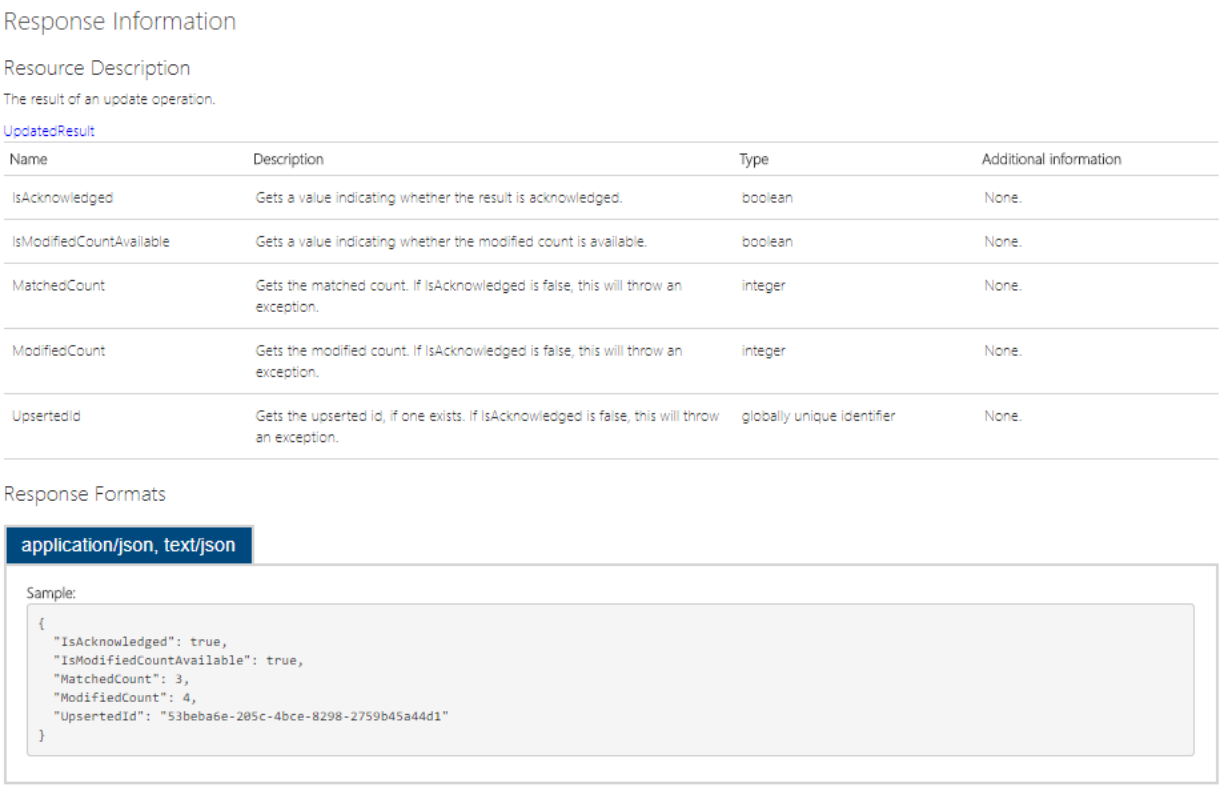

Sl. 4.17. Definicija REST odgovora nakon dohvaćanja sheme

Ažuriranje sheme je treća operacija u radu sa shemama zbirki podataka. Na slici 4.18 prikazana je definicija zahtjeva kojim se izvodi operacija ažuriranja sheme. Za ažuriranje sheme, potreban parametar je naziv sheme i objekt sheme. Naziv sheme se dohvaća iz URL-a zahtjeva, a objekt sheme se dohvaća iz tijela zahtjeva.



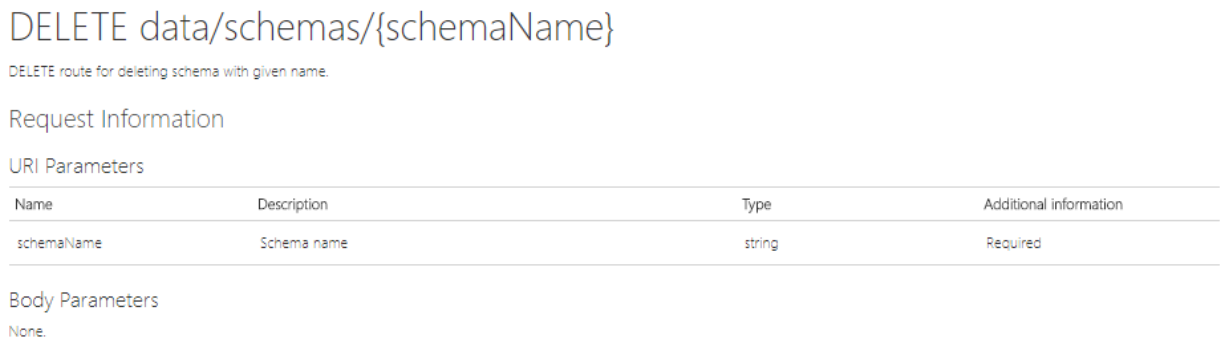
Sl. 4.18. Definicija REST zahtjeva za ažuriranje sheme

Rezultat izvođenja operacije ažuriranja sheme je REST odgovor, čija je definicija prikazana na slici 4.19. U tijelo odgovora smješten je JSON objekt koji prikazuje rezultat operacije ažuriranja. Svojstva objekta su identifikator ažurirane sheme, vrijednost koja indicira uspješnog operacije, vrijednost koja indicira je li dostupan broj ažuriranih shema, broj pronađenih shema i broj ažuriranih shema.



Sl. 4.19. Definicija REST odgovora nakon ažuriranja sheme

Brisanje sheme je četvrta operacija u radu sa shemama zbirki podataka. Na slici 4.20 prikazana je definicija zahtjeva kojim se izvodi operacija brisanja sheme. Za brisanje sheme, potreban parametar je naziv sheme, koji se dohvaća iz URL-a zahtjeva.



Sl. 4.20. Definicija REST zahtjeva za brisanje sheme

Rezultat izvođenja operacije brisanja sheme je REST odgovor, čija je definicija prikazana na slici 4.21. U tijelo odgovora smješten je JSON objekt koji prikazuje rezultat operacije brisanja. Svojstva objekta su broj obrisanih shema i vrijednost koja indicira uspješnost operacije brisanja.

Response Information

Resource Description

The result of an delete operation.

DeletedResult

Name	Description	Type	Additional information
DeletedCount	Gets the deleted count. If IsAcknowledged is false, this will throw an exception.	integer	None.
IsAcknowledged	Gets a value indicating whether the result is acknowledged.	boolean	None.

Response Formats

application/json, text/json

Sample:

```
{
  "DeletedCount": 1,
  "IsAcknowledged": true
}
```

Sl. 4.21. Definicija REST odgovora nakon brisanja sheme

Dohvaćanje sheme prema filtru je peta operacija u radu sa shemama zbirki podataka. Na slici 4.22 prikazana je definicija zahtjeva kojim se izvodi operacija dohvaćanja sheme prema filtru. Za dohvaćanje sheme prema filtru, potrebni parametri su upit za pretraživanje, broj stranica s rezultatima, broj rezultata po stranici i vrijednost koja indicira oblik sortiranja. Svi parametri se dohvaćaju iz URL-a zahtjeva.

GET data/schemas?searchQuery={searchQuery}&page={page}&rpp={rpp}&sort={sort}

GET route for retrieving schemas that matches given query filter.

Request Information

URI Parameters

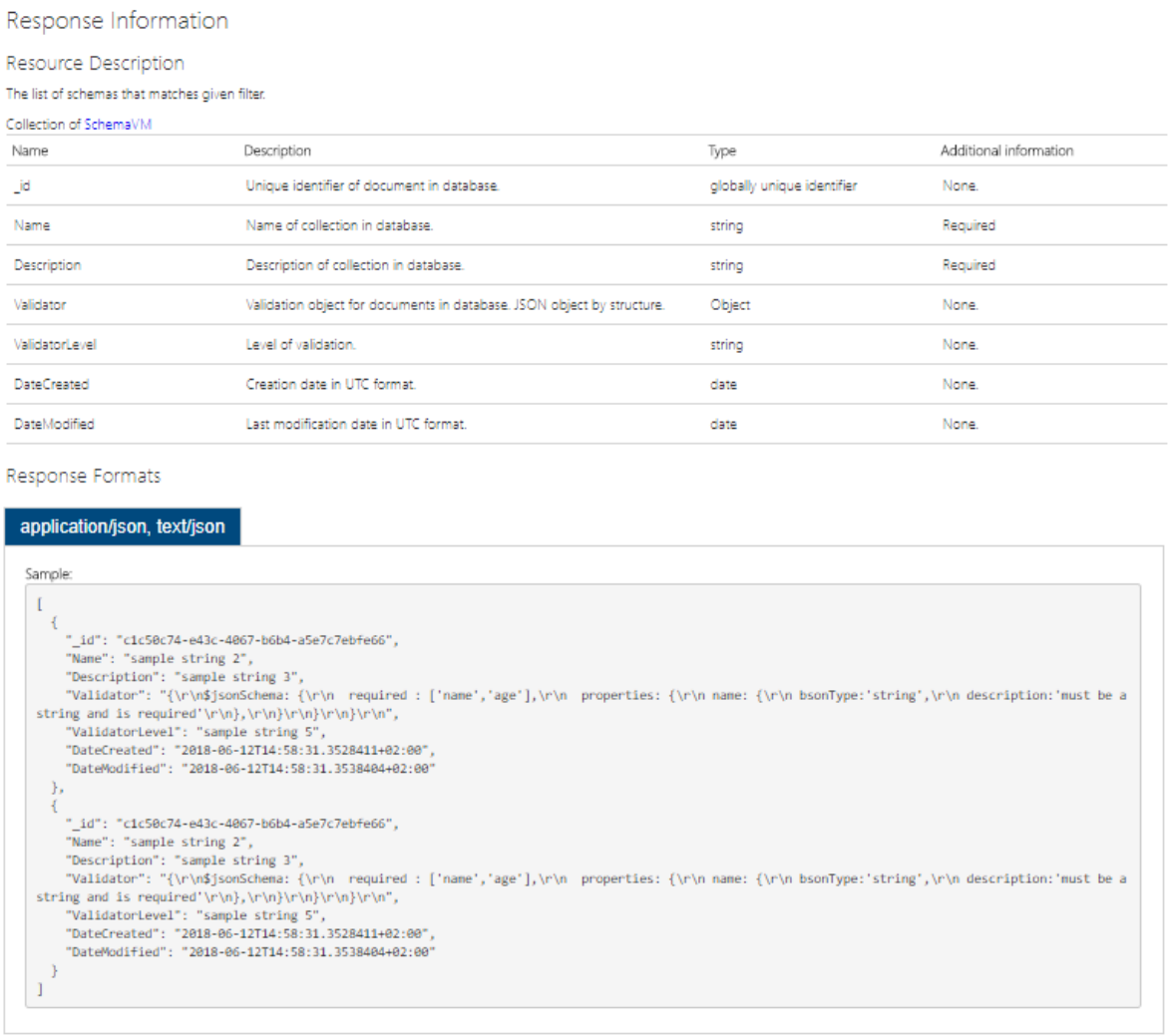
Name	Description	Type	Additional information
searchQuery	Search query value	string	Required
page	Number of result page.	integer	Required
rpp	Number of results per page.	integer	Required
sort	Value that indicates sorting order	string	Required

Body Parameters

None.

Sl. 4.22. Definicija REST zahtjeva za dohvaćanjem shema prema filtru

Rezultat izvođenja operacije dohvaćanja shema prema filtru je REST odgovor, čija je definicija prikazana na slici 4.23. Sheme koje odgovaraju filtru su smještene u tijelo REST odgovora, a predstavljene kao niz JSON objekata.



Response Information

Resource Description

The list of schemas that matches given filter.

Collection of SchemaVM

Name	Description	Type	Additional information
_id	Unique identifier of document in database.	globally unique identifier	None.
Name	Name of collection in database.	string	Required
Description	Description of collection in database.	string	Required
Validator	Validation object for documents in database. JSON object by structure.	Object	None.
ValidatorLevel	Level of validation.	string	None.
DateCreated	Creation date in UTC format.	date	None.
DateModified	Last modification date in UTC format.	date	None.

Response Formats

application/json, text/json

Sample:


```
[
  {
    "_id": "c1c50c74-e43c-4067-b6b4-a5e7c7ebfe66",
    "Name": "sample string 2",
    "Description": "sample string 3",
    "Validator": "{\r\n$jsonSchema: {\r\n  required : ['name','age'],\r\n  properties: {\r\n name: {\r\n bsonType:'string',\r\n description:'must be a string and is required'\r\n},\r\n}\r\n}\r\n}\r\n",
    "ValidatorLevel": "sample string 5",
    "DateCreated": "2018-06-12T14:58:31.3528411+02:00",
    "DateModified": "2018-06-12T14:58:31.3538404+02:00"
  },
  {
    "_id": "c1c50c74-e43c-4067-b6b4-a5e7c7ebfe66",
    "Name": "sample string 2",
    "Description": "sample string 3",
    "Validator": "{\r\n$jsonSchema: {\r\n  required : ['name','age'],\r\n  properties: {\r\n name: {\r\n bsonType:'string',\r\n description:'must be a string and is required'\r\n},\r\n}\r\n}\r\n}\r\n",
    "ValidatorLevel": "sample string 5",
    "DateCreated": "2018-06-12T14:58:31.3528411+02:00",
    "DateModified": "2018-06-12T14:58:31.3538404+02:00"
  }
]
```


Sl. 4.23. Definicija REST odgovora nakon dohvaćanja shema prema filtru

4.3. BaaS usluga upravljanja korisnicima

BaaS usluga upravljanja korisnicima omogućuje klijentu osnovne metode ovjeravanja autentičnosti te upravljanje korisničkim podacima. Upravljanje korisnicima je od ključne važnosti za bilo koju aplikaciju i kao takvo je središte sustava BaaS. Usluga upravljanja korisnicima ima načine za rukovanje tipičnim korisničkim akcijama kao što su registracija, prijava i odjava. Upotreba usluge se ostvaruje putem API-ja koji implementira ovu uslugu. Osim navedenih korisničkih akcija, usluga ima načine za upravljanje korisničkim podacima, kao što su promjena lozinke, promjena i brisanje korisničkog modela i drugi. U ovoj BaaS usluzi, osnova jedinica podataka je korisnik aplikacije, koji je predstavljen domenskim modelom u programskom jeziku C#. Usluga upravljanja korisnicima se dijeli na dvije funkcionalnosti:

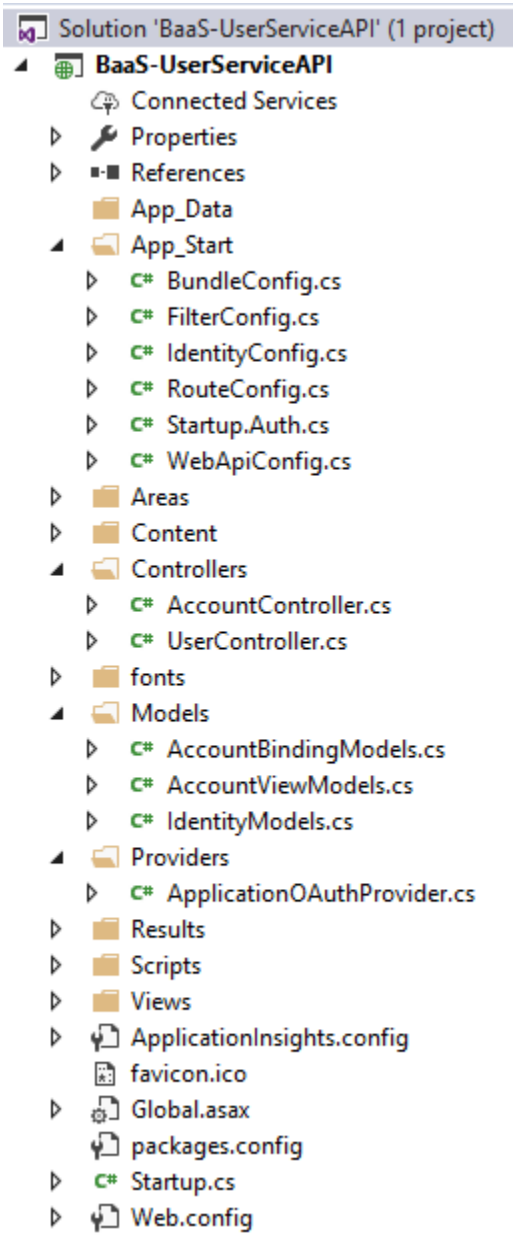
- Ovjeravanje autentičnosti - metode registracije, prijave, odjave, postavljanja i promjene lozinke.
- Upravljanje korisničkim podacima - CRUD operacije nad modelom korisnika.

4.3.1. Struktura usluge upravljanja korisnicima

BaaS usluga upravljanja korisnicima implementirana je kroz razvojni okvir ASP.NET Web API. S obzirom da je je jedna od glavnih funkcionalnosti ove usluge ovjera autentičnosti, prilikom kreiranja programskog rješenja u Visual Studio razvojnom okruženju, odabran je ugrađeni mehanizam unutar okruženja za ovjeru autentičnosti, pod nazivom Individualni računi (engl. *Individual accounts*). Ukoliko su odabrani individualni korisnički računi, web API će automatski biti konfiguriran za upotrebu okvira za ovjeru autentičnosti ASP.NET Identity. U ovoj usluzi, web API kontroleri djeluju kao poslužitelji resursa, po terminologiji protokola OAuth2.

Usluga upravljanja korisnicima, u obliku web API-ja, sadrži dva kontrolera: *AccountController*, koji provodi ovjeru autentičnosti i *UserController*, koji radi s korisničkim podacima. Oba kontrolera sadrže asinkrone funkcije za izvođenje osnovnih CRUD operacija s pripadnim entitetom. Sve funkcije navedenih kontrolera vraćaju odgovor u formatu JSON s pripadnim HTTP statusnim kodovima. Funkcije primaju podatke iz putanje, tijela i zaglavlja zahtjeva.

Na slici 4.24 prikazan je sadržaj programskog rješenja BaaS usluge upravljanja korisnicima. Usluga sadrži tri skupine modela: domenski modeli identiteta korisnika, modeli za prikaz (engl. *view models*) i modeli za vezivanje prethodna dva. Sve skupine modela imaju ekvivalentna svojstva, no predstavljena različitim tipovima podataka ili poslovnom logikom, ovisno za koji specifični dio usluge su vezane. Kontroleri imaju tijesnu ovisnost o objektu upravitelja korisnika, koji je instanca klase *UserManager*, a dohvaća se iz konteksta OWIN poslužitelja. Klasa *UserManager* služi kao pročelje (engl. *facade*) za prijavu i odjavu korisnika. On sadrži osnovne metode za ovjeru autentičnosti i upravljanje korisničkim podacima, ali služi kao dodatna apstrakcija implementacije okvira ASP.NET Identity. *UserManager* je pak vezan za klasu *UserStore*, koja je vezana za bazu podataka u kojoj su smješteni korisnički i ostali podaci ove usluge. U ovoj usluzi se, kao tehnologije vezane za bazu podataka, koriste okvir Entity Framework i baza podataka MSSQL.



Sl. 4.24. Sadržaj programskog rješenja BaaS usluge upravljanja korisnicima

4.3.2. Dizajn baze podataka usluge upravljanja korisnicima

Usluga upravljanja korisnicima je vezana uz sigurnosnu opciju Individualni računi, pa je stoga predložak programskog rješenja usluge razdvojen u kod potreban za operacije ovjere autentičnosti sa svim informacijama pohranjenim u bazi podataka SQL Server. Klasa *UserStore*, potrebna za konstruiranje upravitelja korisnika, pa time i kontrolera *AccountController*, ima ovisnost o klasama iz okvira za objektno-relacijsko mapiranje Entity Framework. Konkretno, riječ je o *IdentityDbContext* klasi, koja pruža sva potrebna mapiranja dostupna iz okvira te svojstva potrebna za upravljanje tablicama identiteta u bazi podataka *SQL Server*. Programski kod usluge koristi

klasu *ApplicationDbContext* koja nasljeđuje *IdentityDbContext* klasu s idejom da se mogu dodati vlastita svojstva za entitete, tablice i sveukupne podatke koje usluga treba zadržati u istoj bazi podataka.

Prema standardnim postavkama, klasa *IdentityDbContext* koristi niz za povezivanje (engl. *connection string*) koji ukazuje na bazu podataka SQL Server. Vrijednost niza za povezivanje nalazi se u datoteci *web.config* te je za promjenu baze podataka potrebno samo promijeniti vrijednost niza za povezivanje u toj datoteci. Na slici 4.25 prikazan je programski kod za povezivanje usluge s bazom podataka. Okvir za objektno-relacijsko mapiranje Entity Framework je kompatibilan s bazom podataka SQL Server i postavljanjem konteksta i svojstava unutar konteksta, moguće je automatski generirati strukturu baze podataka. Definicija konteksta ove usluge, uključuje sve objekte iz okvira za ovjeru autentičnosti ASP.NET Identity, s pripadnim svojstvima. Na slici 4.26 prikazan je dijagram baze podataka usluge upravljanja korisnicima.

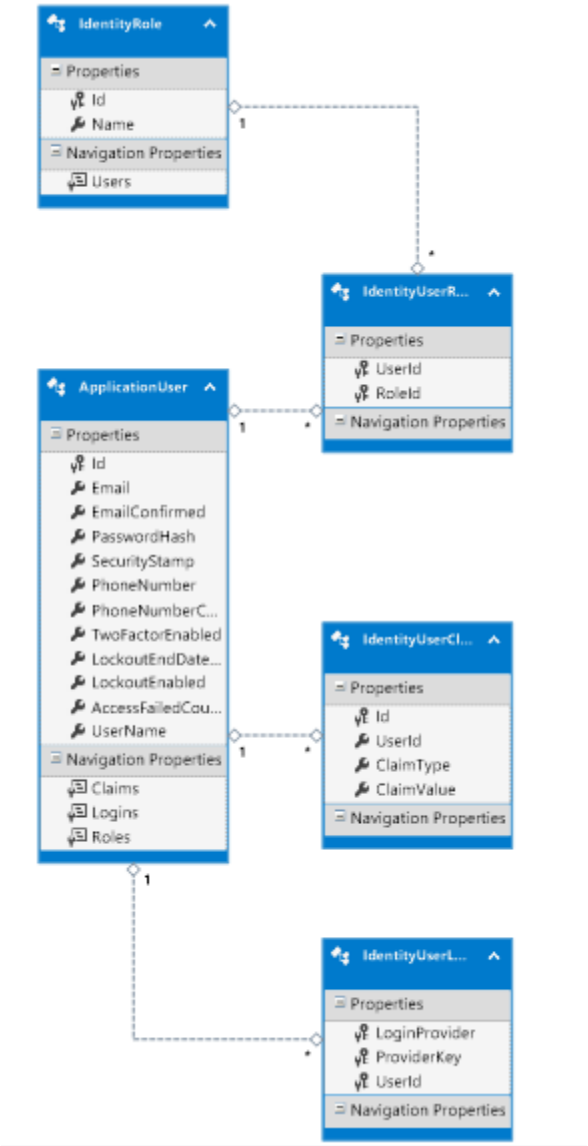
Konstruiranje konteksta za Entity Framework:

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
  {
      public ApplicationDbContext()
          : base("DefaultConnection", throwIfV1Schema: false)
      {
      }

      public static ApplicationDbContext Create()
      {
          return new ApplicationDbContext();
      }
  }
```

Definicija niza za povezivanje u web.config datoteci:

```
<connectionStrings>
    <add name="DefaultConnection" connectionString="Data
Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\aspnet-BaaS-
UserServiceAPI-20180614085901.mdf;Initial Catalog=aspnet-BaaS-UserServiceAPI-
20180614085901;Integrated Security=True"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
```

Sl. 4.25. Programski kod za povezivanje usluge upravljanja korisnicima s bazom podataka



Sl. 4.26. Dijagram baze podataka usluge upravljanja korisnicima

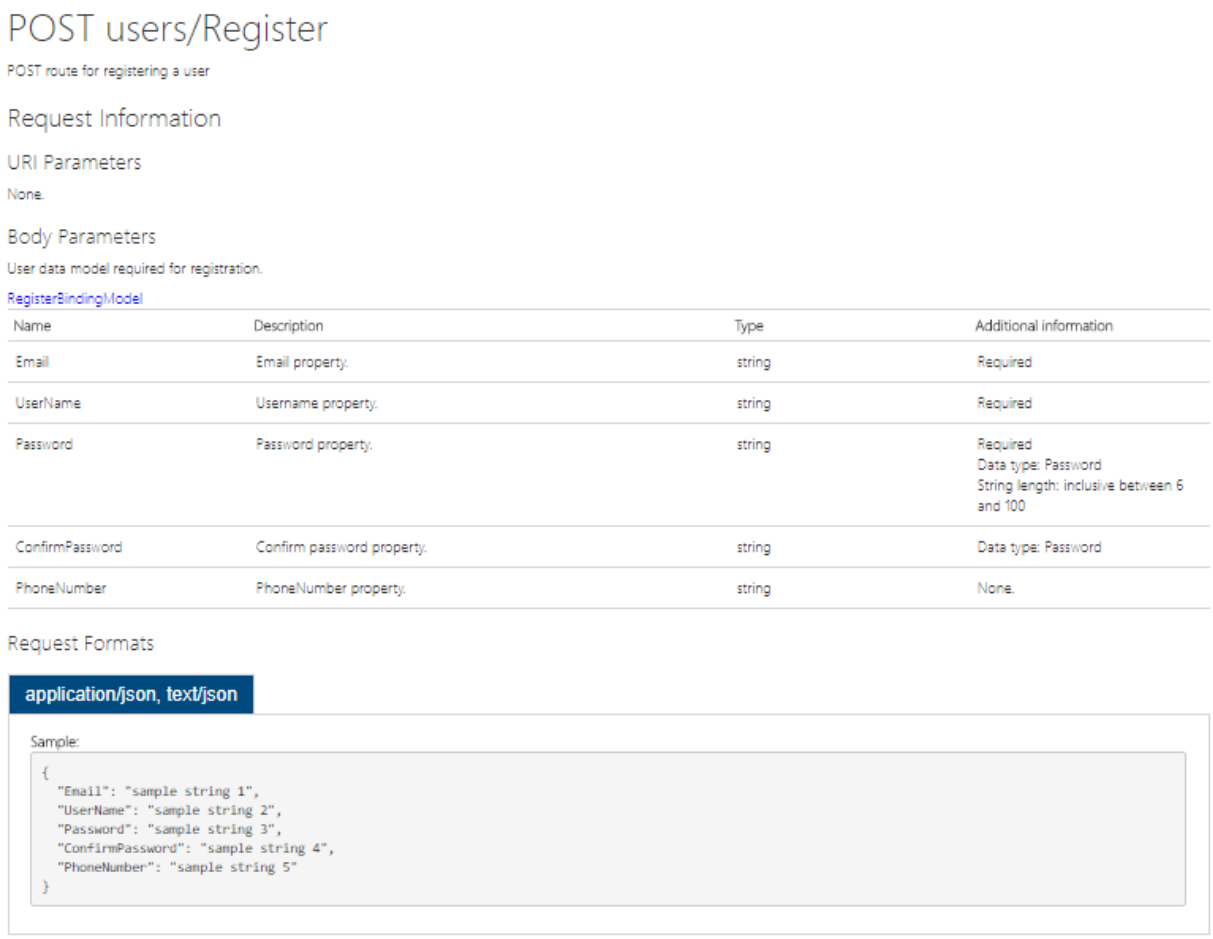
4.3.3. Ovjeravanje autentičnosti

Usluga upravljanja korisnicima kroz okvir web API pruža metode za ovjeru autentičnosti. Autentičnost podrazumijeva poznavanje identiteta korisnika. Funkcionalnost ovjere autentičnosti sadrži tipične operacije za identificiranje korisnika u njegovim akcijama. Te operacije su: registracija, prijava, odjava, postavljanje i promjena lozinke. Svaka operacija se dohvaća putem REST zahtjeva, pri čemu svaki zahtjev mora sadržavati sljedeće parametre:

- Naziv HTTP operacije,
- Krajnju točku ili URL operacije,
- Polje *Content-Type* u zaglavlju postavljeno na *application/json*.

Zahtjevi za operacijama odjave, postavljanja i promjene lozinke, moraju sadržavati i polje *Authorization* u zaglavlju s vrijednosti značke za autorizaciju. Uz navedene obvezne podatke, neke operacije mogu zahtijevati i dodatne parametre.

Registriranje korisnika je prva operacija unutar funkcionalnosti ovjere autentičnosti. Na slici 4.27 prikazana je definicija zahtjeva kojim se izvodi operacija registriranja korisnika. Za registriranje korisnika, potreban parametar je korisnički model, koji se dohvaća iz tijela zahtjeva.



POST users/Register

POST route for registering a user

Request Information

URI Parameters

None.

Body Parameters

User data model required for registration.

RegisterBindingModel

Name	Description	Type	Additional information
Email	Email property.	string	Required
UserName	Username property.	string	Required
Password	Password property.	string	Required Data type: Password String length: inclusive between 6 and 100
ConfirmPassword	Confirm password property.	string	Data type: Password
PhoneNumber	PhoneNumber property.	string	None.

Request Formats

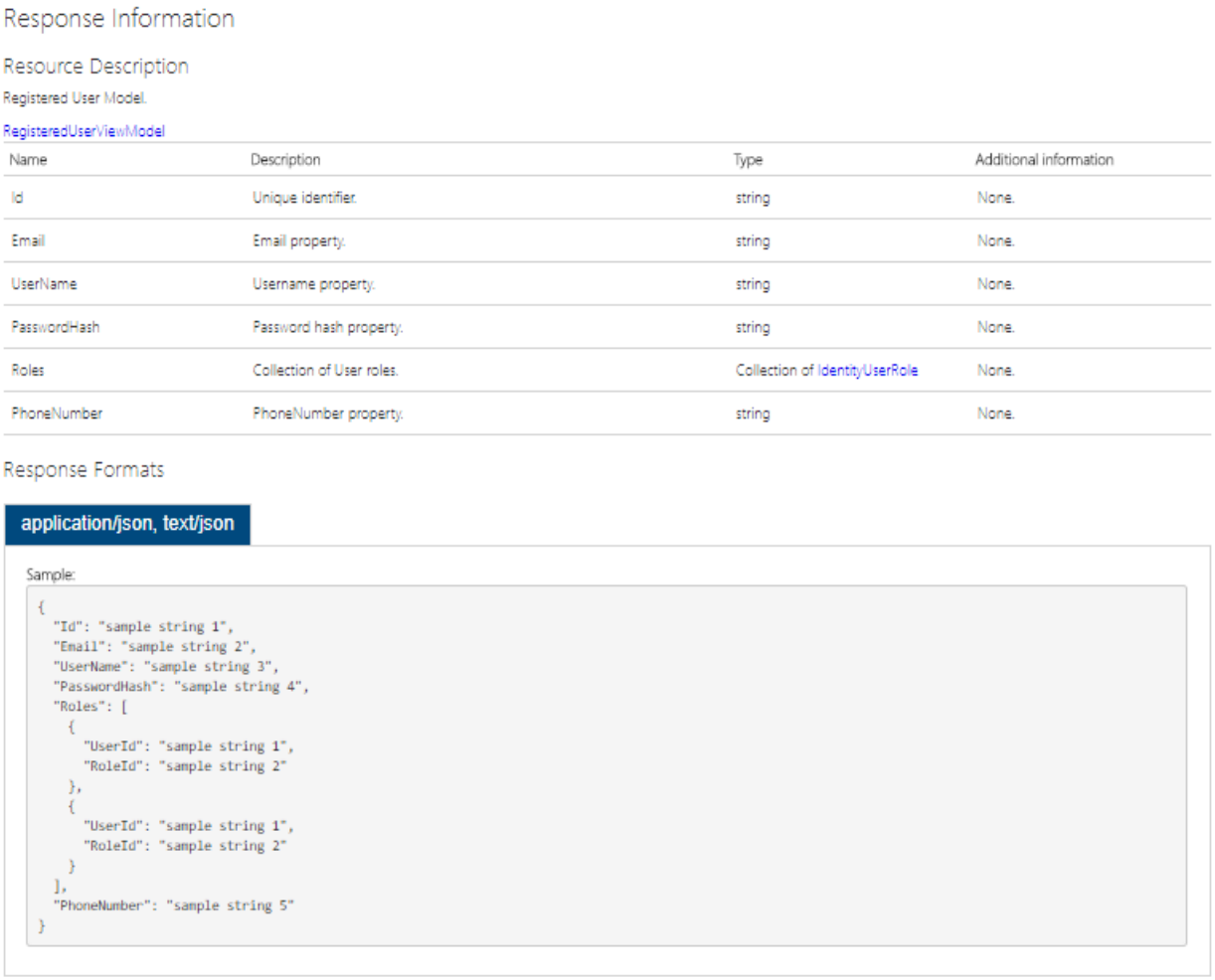
application/json, text/json

Sample:

```
{
  "Email": "sample string 1",
  "UserName": "sample string 2",
  "Password": "sample string 3",
  "ConfirmPassword": "sample string 4",
  "PhoneNumber": "sample string 5"
}
```

Sl. 4.27. Definicija REST zahtjeva za registriranje korisnika

Rezultat izvođenja operacije registriranja korisnika je REST odgovor, čija je definicija prikazana na slici 4.28. Registrirani korisnik je smješten u tijelo REST odgovora, a predstavljen kao JSON objekt. Sadrži novostvorene podatke: identifikator korisnika, kriptiranu lozinku i niz uloga korisnika.



Response Information

Resource Description

Registered User Model.

RegisteredUserViewModel

Name	Description	Type	Additional information
Id	Unique identifier.	string	None.
Email	Email property.	string	None.
UserName	Username property.	string	None.
PasswordHash	Password hash property.	string	None.
Roles	Collection of User roles.	Collection of IdentityUserRole	None.
PhoneNumber	PhoneNumber property.	string	None.

Response Formats

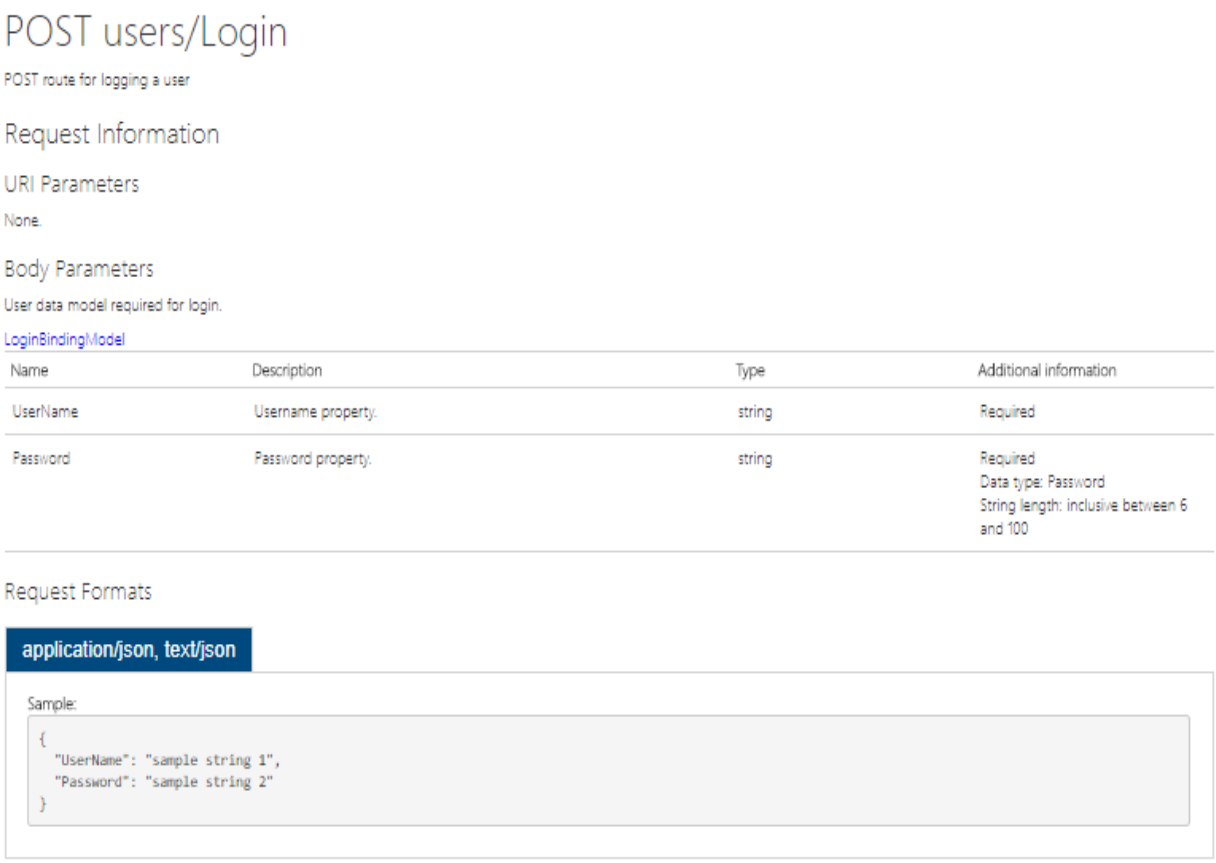
application/json, text/json

Sample:

```
{
  "Id": "sample string 1",
  "Email": "sample string 2",
  "UserName": "sample string 3",
  "PasswordHash": "sample string 4",
  "Roles": [
    {
      "UserId": "sample string 1",
      "RoleId": "sample string 2"
    },
    {
      "UserId": "sample string 1",
      "RoleId": "sample string 2"
    }
  ],
  "PhoneNumber": "sample string 5"
}
```

Sl. 4.28. Definicija REST odgovora nakon registriranja korisnika

Prijava korisnika je druga operacija unutar funkcionalnosti ovjere autentičnosti. Na slici 4.29 prikazana je definicija zahtjeva kojim se izvodi operacija prijave korisnika. Za prijavu korisnika, potreban parametar je model korisnika, koji se dohvaća iz tijela zahtjeva, a sadrži korisničko ime i lozinku.



Sl. 4.29. Definicija REST zahtjeva za prijavu korisnika

Rezultat izvođenja operacije prijave korisnika je REST odgovor, čija je definicija prikazana na slici 4.30. Prijavljeni korisnik je smješten u tijelo REST odgovora, a predstavljen kao JSON objekt. Sadrži novostvorene podatke: identifikator korisnika, kodiranu lozinku i niz uloga korisnika.

Response Information

Resource Description

Logged in User Model.

LoggedInUserViewModel

Name	Description	Type	Additional information
Id	Unique identifier.	string	None.
Email	Email property.	string	None.
UserName	Username property.	string	None.
PasswordHash	Password hash property.	string	None.
Roles	Collection of User roles.	Collection of IdentityUserRole	None.
PhoneNumber	PhoneNumber property.	string	None.

Response Formats

application/json, text/json

Sample:

```
{
  "Id": "sample string 1",
  "Email": "sample string 2",
  "UserName": "sample string 3",
  "PasswordHash": "sample string 4",
  "Roles": [
    {
      "UserId": "sample string 1",
      "RoleId": "sample string 2"
    },
    {
      "UserId": "sample string 1",
      "RoleId": "sample string 2"
    }
  ],
  "PhoneNumber": "sample string 5"
}
```

Sl. 4.30. Definicija REST odgovora nakon prijave korisnika

Odjava korisnika je treća operacija unutar funkcionalnosti ovjere autentičnosti. Na slici 4.31 prikazana je definicija zahtjeva kojim se izvodi operacija odjave korisnika. Za odjavu korisnika, potreban parametar je pristupna značka, koja se dohvaća iz zaglavlja zahtjeva. Posjedovanjem pristupne značke, usluga upravljanja korisnicima raspoznaje koji korisnik zahtjeva odjavu.

POST users/Logout

POST route for logging out a user

Request Information

URI Parameters

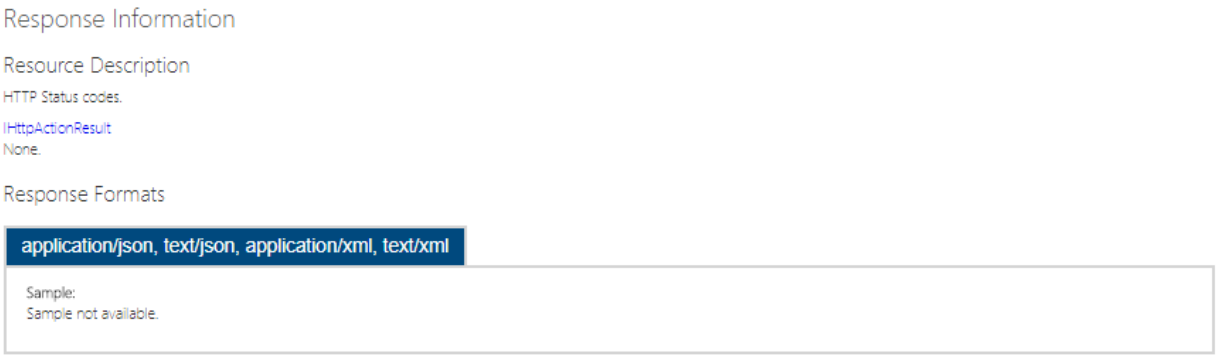
None.

Body Parameters

None.

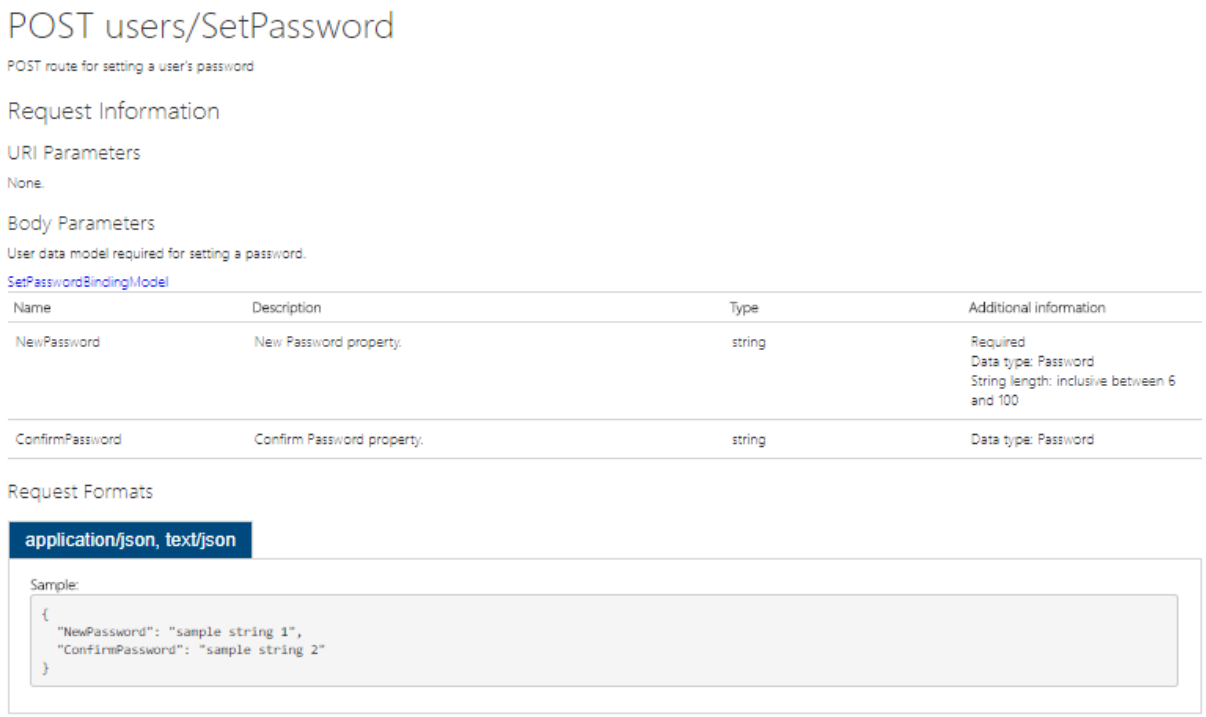
Sl. 4.31. Definicija REST zahtjeva za odjavu korisnika

Rezultat izvođenja operacije odjave korisnika je REST odgovor, čija je definicija prikazana na slici 4.32.



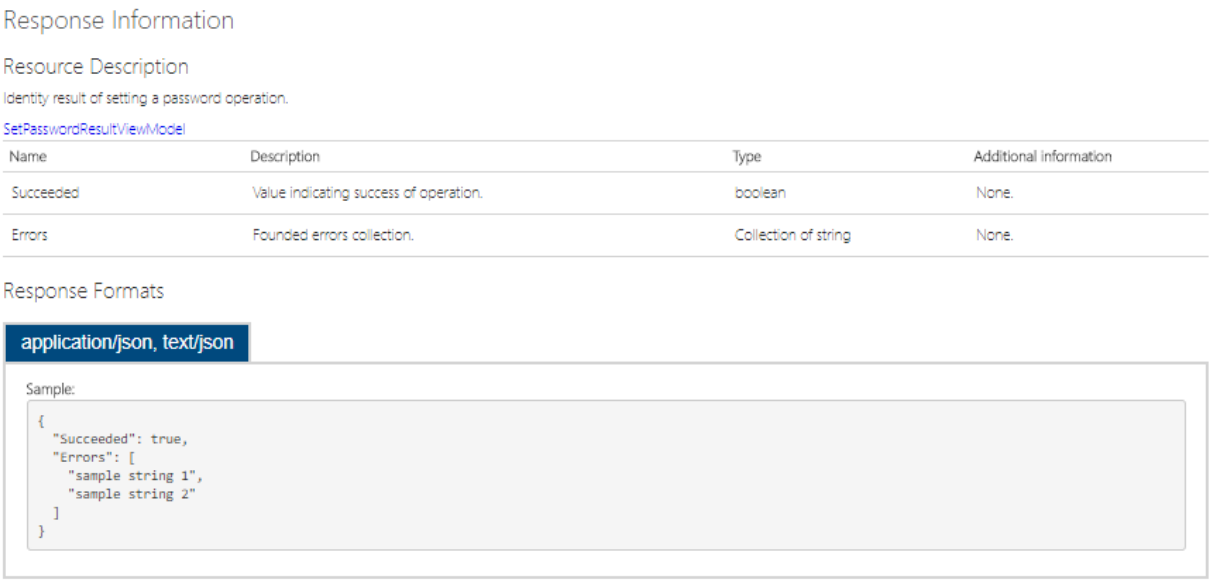
Sl. 4.32. Definicija REST odgovora nakon odjave korisnika

Postavljanje korisničke lozinke je četvrta operacija unutar funkcionalnosti ovjere autentičnosti. Na slici 4.33 prikazana je definicija zahtjeva kojim se izvodi operacija postavljanja lozinke. Za postavljanje lozinke, potrebni parametri su pristupna značka i model korisnika. Pristupna značka se dohvaća iz zaglavlja zahtjeva, a model korisnika se dohvaća iz tijela zahtjeva.



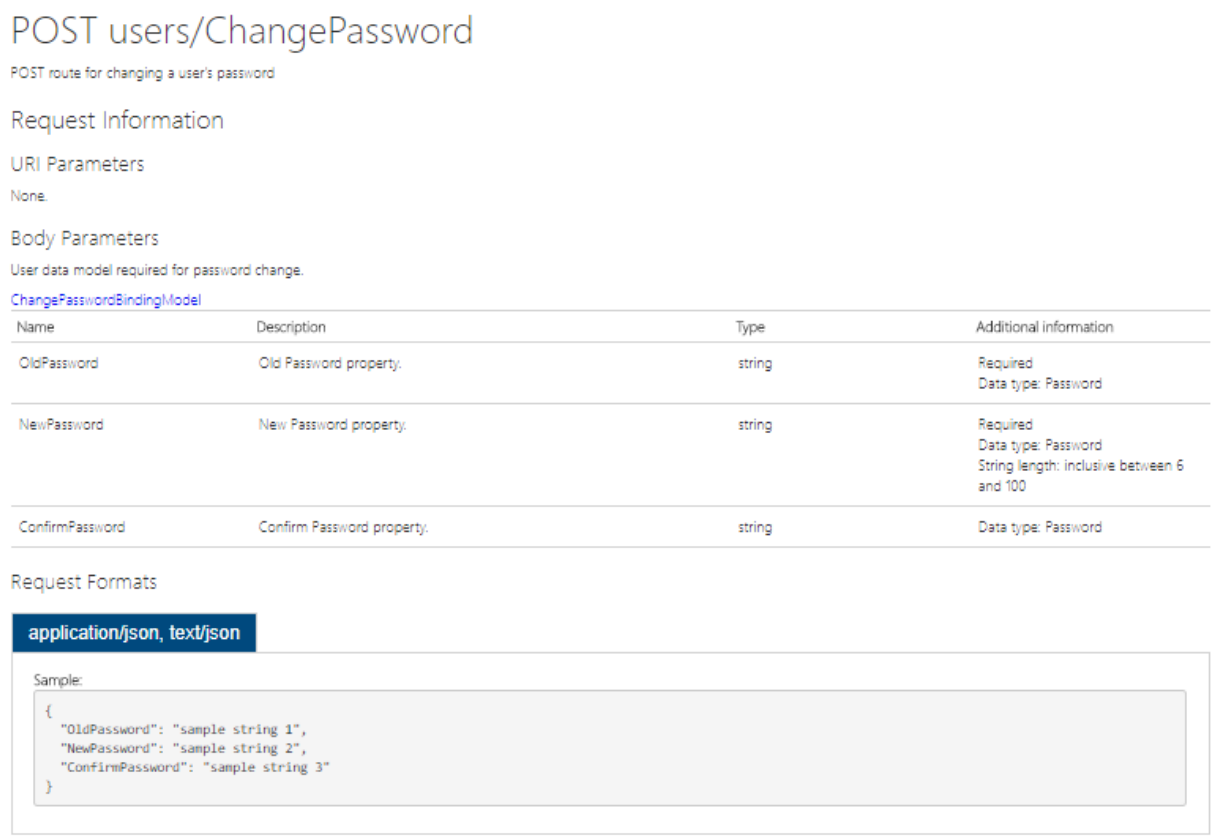
Sl. 4.33. Definicija REST zahtjeva za postavljanje lozinke

Rezultat izvođenja operacije postavljanje lozinke je REST odgovor, čija je definicija prikazana na slici 4.34. U tijelo odgovora smješten je JSON objekt koji prikazuje rezultat operacije postavljanja lozinke. Svojstva objekta su vrijednost koja indicira uspješnost operacije i zbirka počinjenih grešaka.



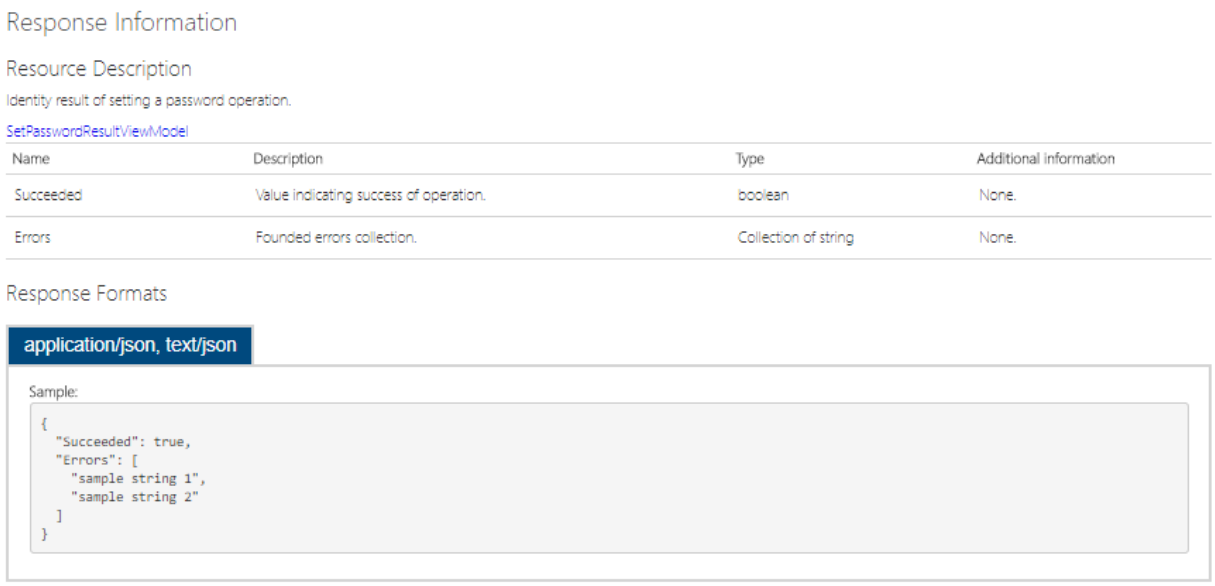
Sl. 4.34. Definicija REST odgovora nakon postavljanja lozinke

Promjena korisničke lozinke je peta operacija unutar funkcionalnosti ovjere autentičnosti. Na slici 4.35 prikazana je definicija zahtjeva kojim se izvodi operacija promjene lozinke. Za promjenu lozinke, potrebni parametri su pristupna značka i model korisnika. Pristupna značka se dohvaća iz zaglavlja zahtjeva, a model korisnika se dohvaća iz tijela zahtjeva.



Sl. 4.35. Definicija REST zahtjeva za promjenu lozinke

Rezultat izvođenja operacije promjene lozinke je REST odgovor, čija je definicija prikazana na slici 4.36. U tijelo odgovora smješten je JSON objekt koji prikazuje rezultat operacije promjene lozinke. Svojstva objekta su vrijednost koja indicira uspješnost operacije i zbirka počinjenih grešaka.



Sl. 4.36. Definicija REST odgovora nakon postavljanja lozinke

4.3.4. Upravljanje korisničkim podacima

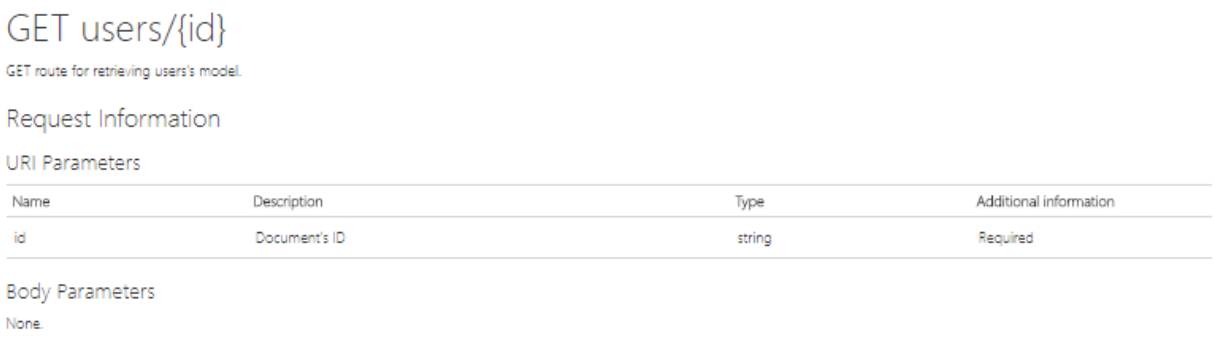
Usluga upravljanja korisnicima kroz okvir web API pruža metode za upravljanje korisničkim podacima. Funkcionalnost upravljanja korisničkim podacima sadrži tipične CRUD operacije nad modelom korisnika. Te operacije su: dohvaćanje korisničkog modela, ažuriranje korisničkog modela, brisanje korisnika i pretraživanje tablice korisnika prema zadanom filtru. Svaka operacija se dohvaća putem REST zahtjeva, pri čemu svaki zahtjev mora sadržavati sljedeće parametre:

- Naziv HTTP operacije,
- Krajnju točku ili URL operacije,
- Polje *Authorization* u zaglavlju s vrijednosti značke za autorizaciju,
- Polje *Content-Type* u zaglavlju postavljeno na *application/json*.

Uz navedene obvezne podatke, neke operacije mogu zahtijevati i dodatne parametre.

Dohvaćanje korisničkog modela je prva operacija unutar funkcionalnosti upravljanja korisničkim podacima. Na slici 4.37 prikazana je definicija zahtjeva kojim se izvodi operacija dohvaćanja

korisničkog modela. Za dohvaćanje korisničkog modela, potreban parametar je identifikator korisnika, koji se dohvaća iz URL-a zahtjeva.



Sl. 4.37. Definicija REST zahtjeva za dohvaćanje korisničkog modela

Rezultat izvođenja operacije dohvaćanja korisničkog modela je REST odgovor, čija je definicija prikazana na slici 4.38. Traženi model je smješten u tijelo REST odgovora, a predstavljen kao JSON objekt. Svojstva modela su e-mail korisnika, kriptirana lozinka, sigurnosni pečat, broj telefona, vrijednost koja indicira dvostruku faktorsku sigurnost, datum zaključavanja i vrijednost koja indicira zaključavanje korisnika, broj neuspješnih prijava, niz korisničkih uloga, zahtjeva i prijava, identifikator korisnika i korisničko ime.

Sl. 4.38. Definicija REST odgovora nakon dohvaćanja korisničkog modela

Ažuriranje korisničkog modela je druga operacija unutar funkcionalnosti upravljanja korisničkim podacima. Na slici 4.39 prikazana je definicija zahtjeva kojim se izvodi operacija ažuriranja korisničkog modela. Za ažuriranje korisničkog modela, potrebni parametri su identifikator

korisnika i korisnički model. Identifikator resursa se dohvaća iz URL-a zahtjeva, a korisnički model iz tijela zahtjeva.

PUT users/{id}

PUT route for updating user's model.

Request Information

URI Parameters

Name	Description	Type	Additional information
id	User's ID	globally unique identifier	Required

Body Parameters

User's model required for updating.

RegisterBindingModel

Name	Description	Type	Additional information
Email	Email property.	string	Required
UserName	Username property.	string	Required
Password	Password property.	string	Required Data type: Password String length: inclusive between 6 and 100
ConfirmPassword	Confirm password property.	string	Data type: Password
PhoneNumber	PhoneNumber property.	string	None.

Sl. 4.39. Definicija REST zahtjeva za ažuriranjem korisničkog modela

Rezultat izvođenja operacije ažuriranja korisničkog modela je REST odgovor, čija je definicija prikazana na slici 4.40. U tijelo odgovora smješten je JSON objekt koji prikazuje rezultat operacije ažuriranja. Svojstva objekta su vrijednost koja indicira uspješnost operacije i zbirka počinjenih grešaka.

Response Information

Resource Description

The result of an update operation.

UpdateUserResultViewModel

Name	Description	Type	Additional information
Succeeded	Value indicating success of operation.	boolean	None.
Errors	Founded errors collection.	Collection of string	None.

Response Formats

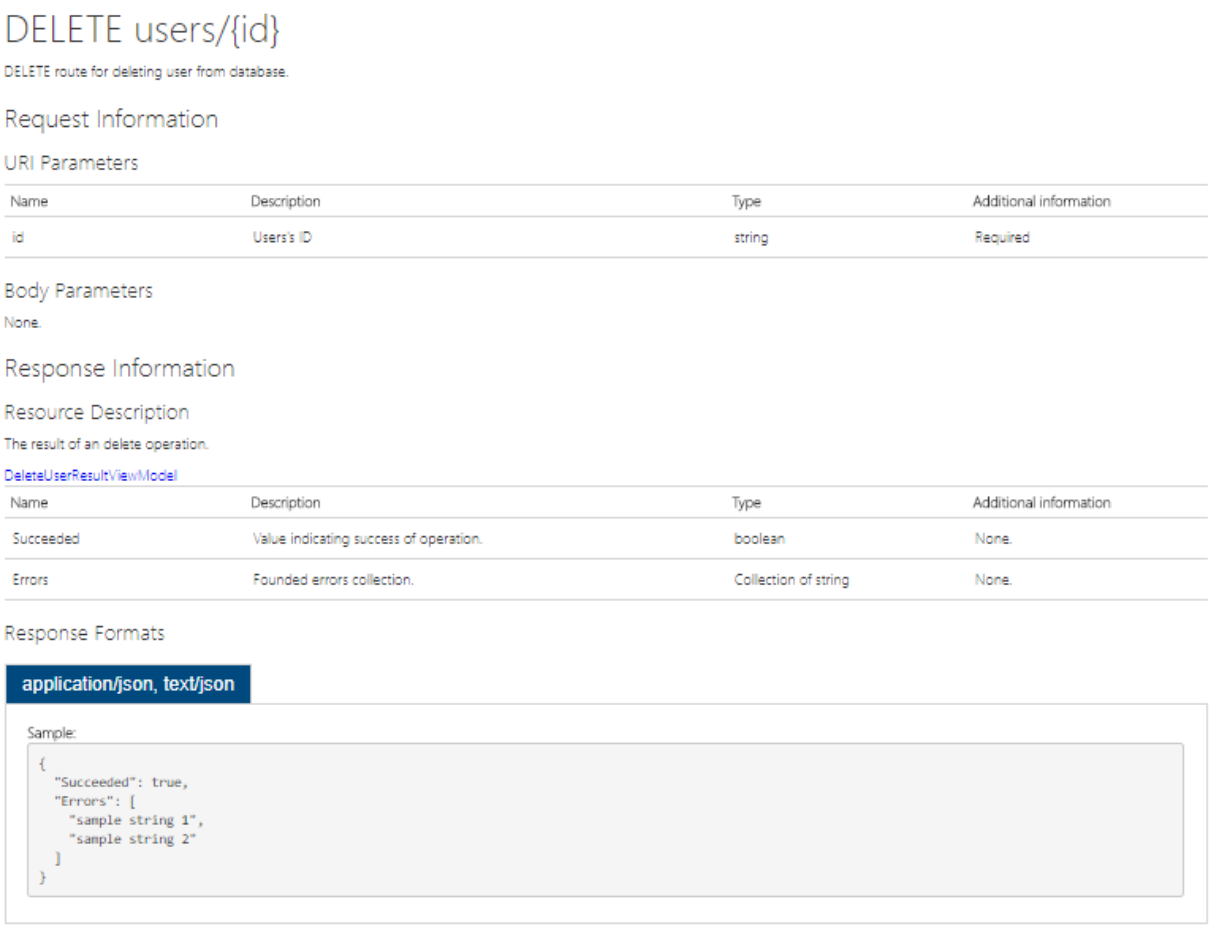
application/json, text/json

Sample:

```
{
  "Succeeded": true,
  "Errors": [
    "sample string 1",
    "sample string 2"
  ]
}
```

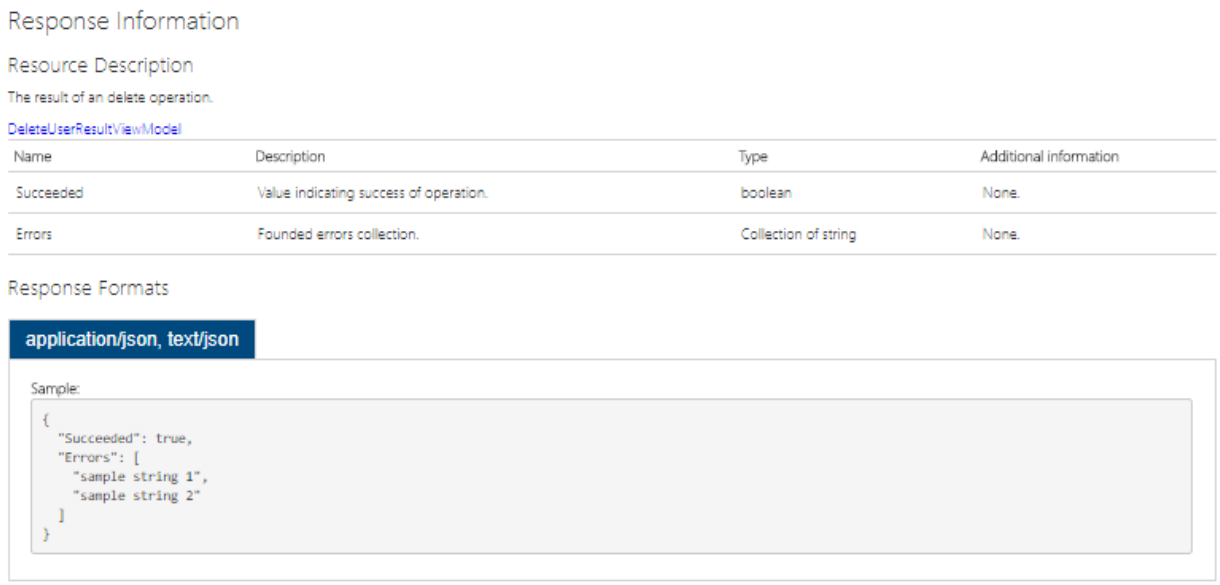
Sl. 4.40. Definicija REST odgovora nakon ažuriranja korisničkog modela

Brisanje korisničkog modela je treća operacija unutar funkcionalnosti upravljanja korisničkim podacima. Na slici 4.41 prikazana je definicija zahtjeva kojim se izvodi operacija brisanja korisničkog modela. Za brisanje korisničkog modela, potreban parametar je identifikator korisnika, koji se dohvaća iz URL-a zahtjeva.



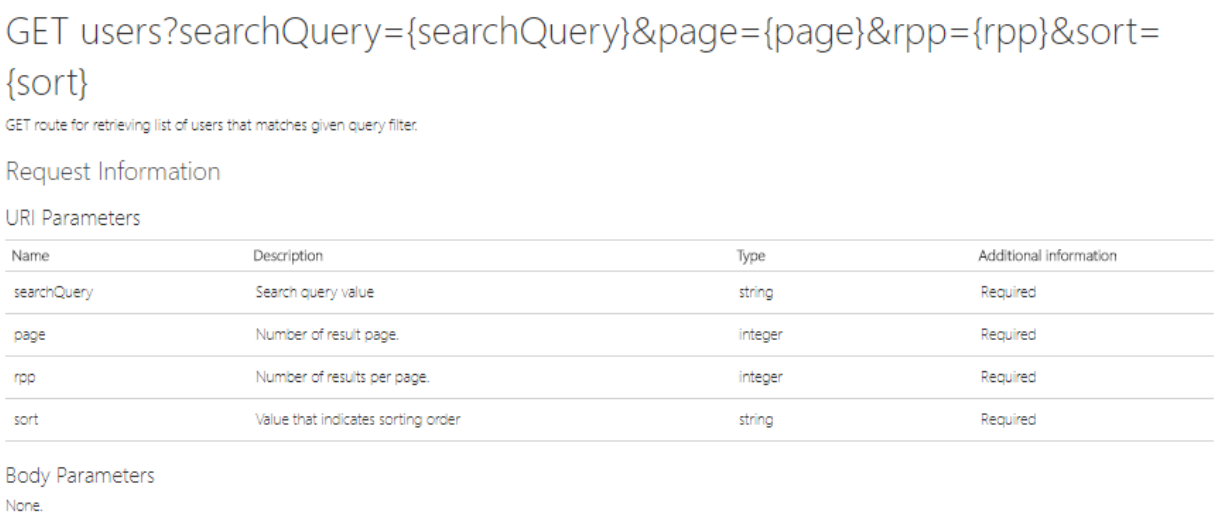
Sl. 4.41. Definicija REST zahtjeva za brisanjem korisničkog modela

Rezultat izvođenja operacije brisanja korisničkog modela je REST odgovor, čija je definicija prikazana na slici 4.42. U tijelo odgovora smješten je JSON objekt koji prikazuje rezultat operacije brisanja. Svojstva objekta su vrijednost koja indicira uspješnost operacije i zbirka počinjenih grešaka.



Sl. 4.42. Definicija REST odgovora nakon brisanja korisničkog modela

Dohvaćanje korisničkih modela prema filtru je četvrta operacija unutar funkcionalnosti upravljanja korisničkim podacima. Na slici 4.43 prikazana je definicija zahtjeva kojim se izvodi operacija dohvaćanja korisničkih modela prema filtru. Za dohvaćanje korisničkih modela prema filtru, potrebni parametri su upit za pretraživanje, broj stranica s rezultatima, broj rezultata po stranici i vrijednost koja indicira oblik sortiranja. Svi parametri se dohvaćaju iz URL-a zahtjeva.



Sl. 4.43. Definicija REST zahtjeva za dohvaćanjem korisničkih modela prema filtru

Rezultat izvođenja operacije dohvaćanja korisničkih modela prema filtru je REST odgovor, čija je definicija prikazana na slici 4.44. Korisnički modeli koji odgovaraju filtru su smješteni u tijelo REST odgovora, a predstavljeni kao niz JSON objekata.

Response Formats

application/json, text/json

Sample:


```
[
  {
    "Email": "sample string 1",
    "EmailConfirmed": true,
    "PasswordHash": "sample string 3",
    "SecurityStamp": "sample string 4",
    "PhoneNumber": "sample string 5",
    "PhoneNumberConfirmed": true,
    "TwoFactorEnabled": true,
    "LockoutEndDateUtc": "2018-06-19T10:43:35.907978+02:00",
    "LockoutEnabled": true,
    "AccessFailedCount": 9,
    "Roles": [],
    "Claims": [],
    "Logins": [],
    "Id": "sample string 10",
    "UserName": "sample string 11"
  },
  {
    "Email": "sample string 1",
    "EmailConfirmed": true,
    "PasswordHash": "sample string 3",
    "SecurityStamp": "sample string 4",
    "PhoneNumber": "sample string 5",
    "PhoneNumberConfirmed": true,
    "TwoFactorEnabled": true,
    "LockoutEndDateUtc": "2018-06-19T10:43:35.907978+02:00",
    "LockoutEnabled": true,
    "AccessFailedCount": 9,
    "Roles": [],
    "Claims": [],
    "Logins": [],
    "Id": "sample string 10",
    "UserName": "sample string 11"
  }
]
```

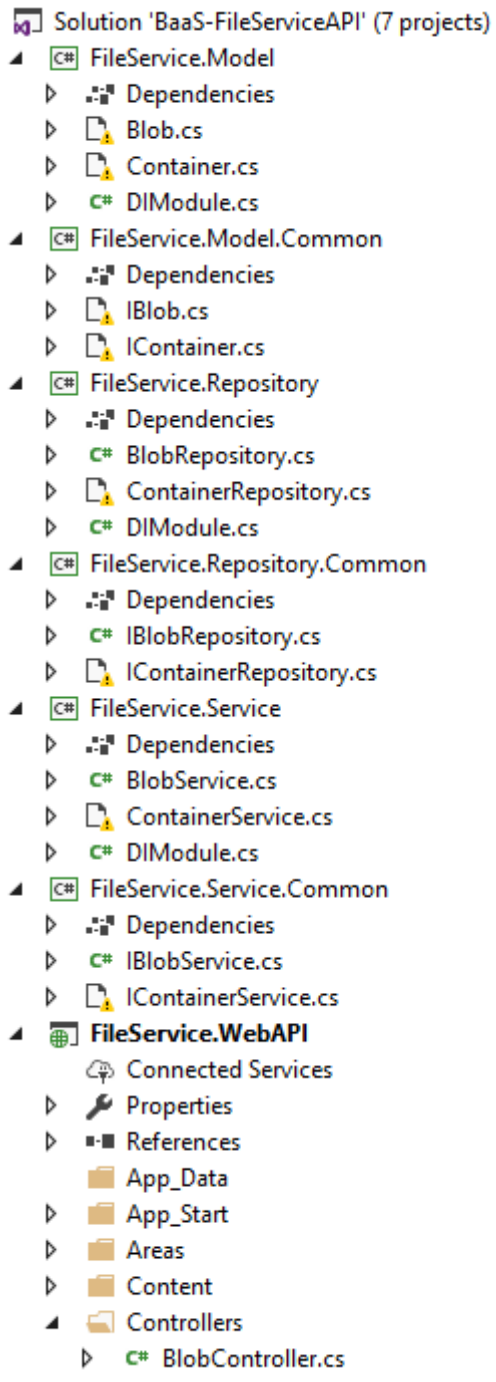

Sl. 4.44. Definicija REST odgovora nakon dohvaćanja korisničkih modela prema filtru

4.4. BaaS usluga upravljanja datotekama

BaaS usluga upravljanja datotekama dodjeljuje namjenski prostor za pohranu datoteka. Pohrana datoteka može se koristiti za pohranu aplikacijskih datoteka i videozapisa na zahtjev. Usluga upravljanja datotekama izlaže API koji omogućuje različite funkcionalnosti u okviru rada s datotekama. Ova BaaS usluga ne služi niti prihvaća datoteke izravno, nego koristi uslugu treće strane za fizičku pohranu. U ovoj usluzi, odabrano je rješenje za pohranu objekata u oblaku Azure Blob Storage. To rješenje je odabrano prvenstveno zbog kompatibilnosti s ostalim tehnologijama, korištenim u razvoju ove usluge. Osim toga, rješenje Azure Blob Storage omogućuje pohranu nestrukturiranih datoteka, što je u skladu sa zahtjevima sustava BaaS, koji mora pružiti što veću fleksibilnost klijentu. Glavna funkcionalnost ove usluge je rad s datotekama. Operacije sadržane unutar funkcionalnosti su: prijenos datoteke, preuzimanje datoteke, promjena imena datoteke i brisanje datoteke.

4.4.1. Struktura usluge upravljanja datotekama

BaaS usluga upravljanja datotekama implementirana je kroz razvojni okvir ASP.NET Web API. Usluga je dizajnirana na temelju višeslojne arhitekture *Onion*, na način da je usluga podijeljena na četiri sloja. Slika 4.45 prikazuje sadržaj programskog rješenja BaaS usluge upravljanja datotekama u obliku skupa projekata razvijenih na platformi .NET. Aplikacijski sloj usluge je web API, a preostala tri sloja su: sloj usluga, sloj repozitorija i sloj modela. Potonja tri sloja su prema vrsti projekta biblioteke klasa.



Sl. 4.45. Sadržaj programskog rješenja BaaS usluge upravljanja datotekama

Aplikacijski sloj BaaS usluge upravljanja datotekama, u obliku web API-ja, sadrži kontroler *BlobController,* koji radi s datotekama. Kontroler sadrži asinkrone funkcije za izvođenje četiriju osnovnih HTTP operacija. Sve funkcije kontrolera vraćaju REST odgovor s pripadnim HTTP statusnim kodovima. Funkcije primaju podatke iz putanje, tijela i zaglavlja zahtjeva.

4.4.2. Rad s datotekama

Osnovna jedinica podataka unutar usluge upravljanja datotekama je datoteka. Datoteka je unutar web API-ja predstavljena domenskim modelom koristeći tipove podataka iz programskog jezika C#. Prilikom komunikacije s bazom podataka, izvodi se mapiranje modela datoteke, u poseban tip podatka naziva *CloudBlockBlob*, koji predstavlja objekt u rješenju za pohranu objekata Azure Blob Storage. Usluga upravljanja datotekama kroz okvir web API pruža metode za rad s nestrukturiranim datotekama. Cjelokupni set CRUD (engl. *Create, Read, Update and Delete*) operacija dostupan je za manipuliranje datotekama. Svaka operacija se dohvaća putem REST zahtjeva, pri čemu svaki zahtjev mora sadržavati sljedeće parametre:

- Naziv HTTP operacije,
- Krajnju točku ili URL operacije,
- Polje *Authorization* u zaglavlju s vrijednosti značke za autorizaciju.

Uz navedene obvezne podatke, neke operacije mogu zahtijevati i dodatne parametre.

Prijenos (engl. *upload*) datoteke je prva operacija u radu s datotekama. Na slici 4.46 prikazana je definicija zahtjeva kojim se izvodi operacija prijenosa datoteke. Za prijenos datoteke, potrebni parametri su ime datoteke i sama kodirana datoteka. Ime datoteke se dohvaća iz URL-a zahtjeva, a datoteka iz tijela zahtjeva. S obzirom da datoteka koja se prenosi, ne mora biti binarna datoteka, potrebno je postaviti polje *Content-Type* u zaglavlju na *multipart/form-data*.

POST files/{fileName}

POST route for uploading a file.

Request Information

URI Parameters

Name	Description	Type	Additional information
fileName	Filename	string	Required

Body Parameters

File stream

HttpPostedFileBase

Name	Description	Type	Additional information
ContentLength		integer	None.
ContentType		string	None.
FileName		string	None.
InputStream		Stream	None.

Sl. 4.46. Definicija REST zahtjeva za prijenos datoteke

Rezultat izvođenja operacije prijenosa datoteke je REST odgovor, čija je definicija prikazana na slici 4.47. U tijelo odgovora je smješten URL preko kojeg se može izvršiti preuzimanje datoteke.

Response Information

Resource Description

Path to created file.

string

Response Formats

application/json, text/json

Sample:

```
"sample string 1"
```

Sl. 4.47. Definicija REST odgovora nakon prijenosa datoteke

Preuzimanje (engl. *download*) datoteke je druga operacija u radu s datotekama. Na slici 4.48 prikazana je definicija zahtjeva kojim se izvodi operacija preuzimanja datoteke. Za preuzimanje datoteke, potreban parametar je ime datoteke, koji se dohvaća iz URL-a zahtjeva.

GET files/{fileName}

GET route for downloading a file.

Request Information

URI Parameters

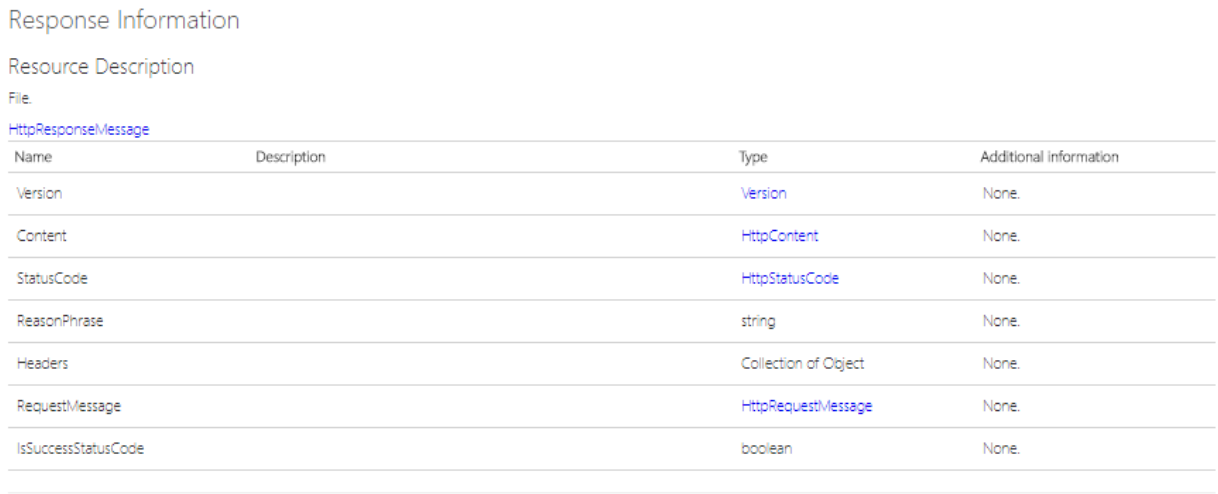
Name	Description	Type	Additional information
fileName	Filename	string	Required

Body Parameters

None.

Sl. 4.48. Definicija REST zahtjeva za preuzimanje datoteke

Rezultat izvođenja operacije preuzimanja datoteke je REST odgovor, čija je definicija prikazana na slici 4.49. U sadržaj odgovora je smještena binarno kodirana datoteka, a zaglavlja odgovora upućuju na vrstu datoteke.



Response Information

Resource Description

File.

HttpResponseMessage

Name	Description	Type	Additional information
Version		Version	None.
Content		HttpContent	None.
StatusCode		HttpStatusCode	None.
ReasonPhrase		string	None.
Headers		Collection of Object	None.
RequestMessage		HttpRequestMessage	None.
IsSuccessStatusCode		boolean	None.

Sl. 4.49. Definicija REST odgovora nakon preuzimanja datoteke

Promjena imena datoteke je treća operacija u radu s datotekama. Na slici 4.50 prikazana je definicija zahtjeva kojim se izvodi operacija promjene imena datoteke. Za promjenu imena datoteke, potrebni parametri su ime datoteke i novo ime datoteke. Ime datoteke se dohvaća iz URL-a zahtjeva, a novo ime iz tijela zahtjeva.

PUT files/{fileName}

PUT route for renaming a file.

Request Information

URI Parameters

Name	Description	Type	Additional information
fileName	Filename	string	Required

Body Parameters

New filename.

string

Request Formats

application/json, text/json

Sample:

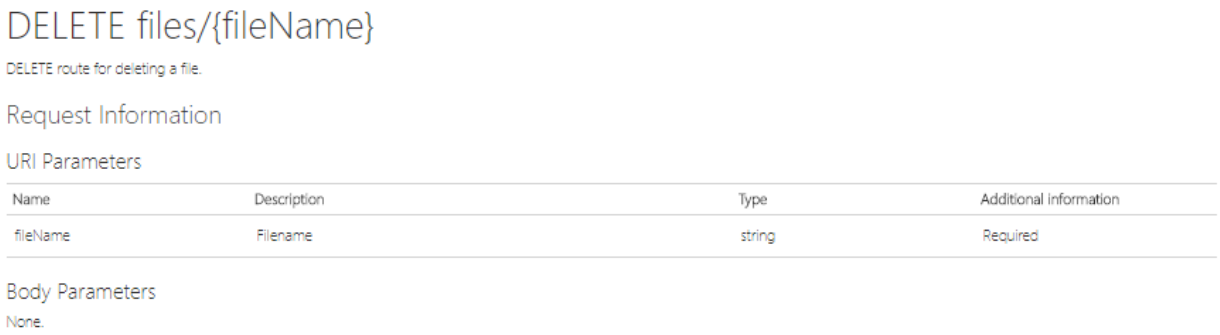
```
"sample string 1"
```

Sl. 4.50. Definicija REST zahtjeva za promjenom imena datoteke

Rezultat izvođenja operacije promjena imena datoteke je REST odgovor, čija je definicija prikazana na slici 4.51. U tijelo odgovora je smješten URL preko kojeg se može izvršiti preuzimanje datoteke.

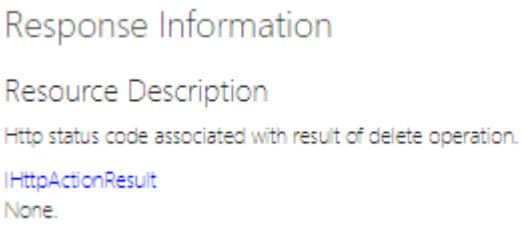
Sl. 4.51. Definicija REST odgovora nakon promjene imena datoteke

Brisanje datoteke je četvrta operacija u radu s datotekama. Na slici 4.52 prikazana je definicija zahtjeva kojim se izvodi operacija brisanja datoteke. Za brisanje datoteke, potreban parametar je ime datoteke, koji se dohvaća iz URL-a zahtjeva.



Sl. 4.52. Definicija REST zahtjeva za brisanjem datoteke

Rezultat izvođenja operacije brisanja datoteke je REST odgovor, čija je definicija prikazana na slici 4.53. Odgovor nema sadržaj, no prema statusnim kodovima se određuje uspješnost operacije brisanja.



Sl. 4.53. Definicija REST odgovora nakon brisanja datoteke

4.5. API vrata sustava BaaS

API vrata (engl. *API gateway*) su u suštini API koji predstavlja jedinstvenu ulaznu točku za sve klijente. Prilikom razvijanja programskog rješenja na temelju arhitekture mikrousluga, API vrata

su jedan od temeljnih uzoraka dizajna (engl. *design patterns*) [36]. API vrata obrađuju zahtjeve na jedan od dva načina. Prvi način je da se zahtjev proslijedi odgovarajućoj usluzi, a drugi način podrazumijeva raspoređivanje obrade zahtjeva na više usluga. API vrata razvijenog sustava BaaS sadrže dvije funkcionalnosti:

- Upravljanje aplikacijama,
- Filtriranje i raspoređivanje prometa.

4.5.1. Upravljanje aplikacijama

Sustav BaaS mora imati mogućnost posluživanja više od jedne aplikacije istovremeno. Klijentska aplikacija koristi usluge sustava BaaS podnoseći pravilno strukturirane zahtjeve. Svaki zahtjev mora sadržavati javni ključ aplikacije u URL-u zahtjeva. Klijent stječe javni ključ aplikacije izvodeći operaciju registriranja aplikacije unutar funkcionalnosti upravljanja aplikacijama. Oblik REST zahtjeva za neku od operacija, triju usluga sustava BaaS, je prikazan na slici 4.54.

Usluga pohrane podataka:

{domena sustava BaaS}/{javni ključ aplikacije}/data/

Usluga upravljanja korisnicima:

{domena sustava BaaS}/{javni ključ aplikacije}/users/

Usluga upravljanja datotekama:

{domena sustava BaaS}/{javni ključ aplikacije}/files/

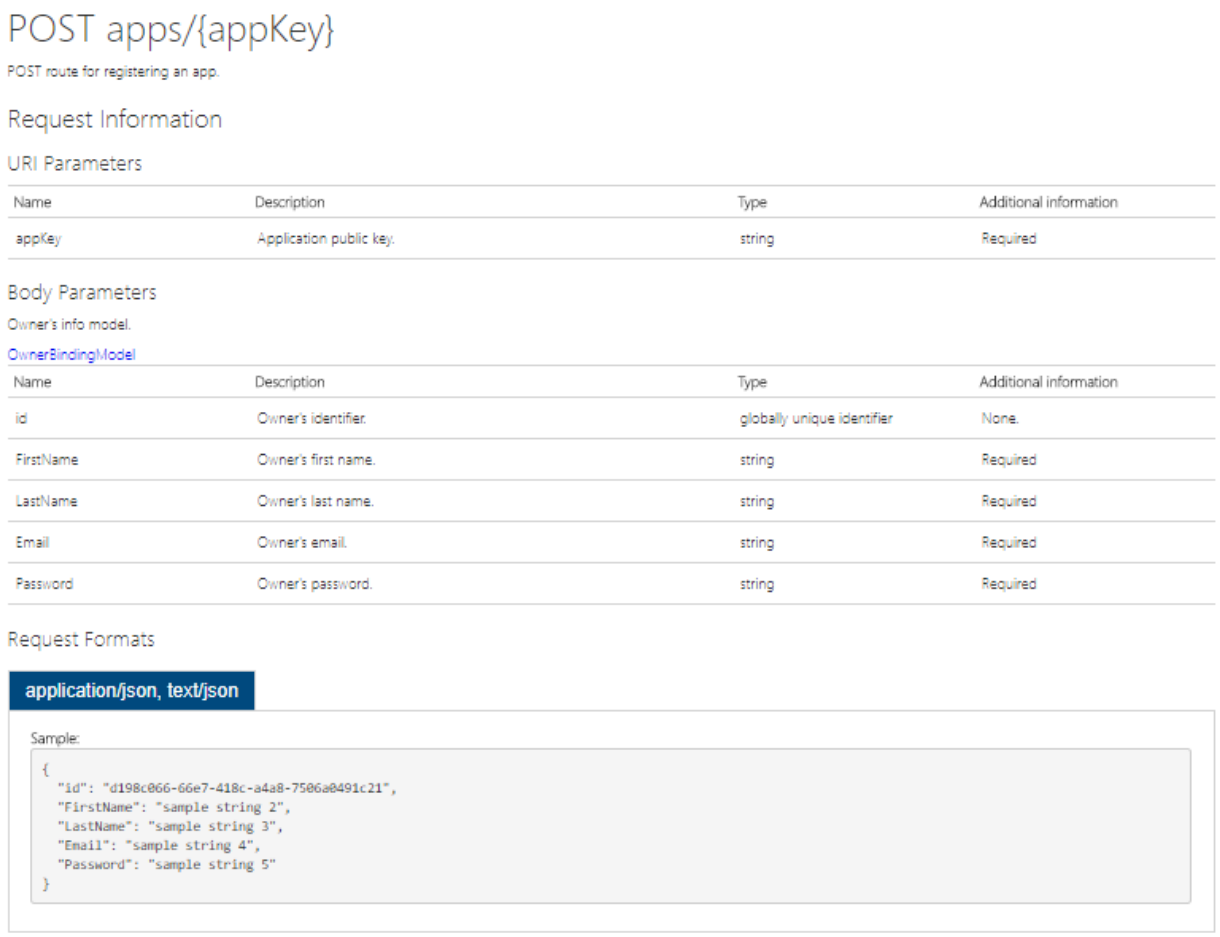
Sl. 4.54. Struktura REST zahtjeva za usluge sustava BaaS

API vrata sustava BaaS kroz okvir web API pružaju metode za upravljanje aplikacijama. Instanca aplikacije je unutar web API-ja predstavljena domenskim modelom koristeći tipove podataka iz programskog jezika C#. Cjelokupni set CRUD operacija dostupan je za upravljanje aplikacijama. Svaka operacija se dohvaća putem REST zahtjeva, pri čemu svaki zahtjev mora sadržavati sljedeće parametre:

- Naziv HTTP operacije,
- Krajnju točku ili URL operacije,
- Polje *Content-Type* u zaglavlju postavljeno na *application/json*.

Uz navedene obvezne podatke, neke operacije mogu zahtijevati i dodatne parametre.

Registriranje aplikacije je prva operacija unutar funkcionalnosti upravljanja aplikacijama. Na slici 4.55 prikazana je definicija zahtjeva kojim se izvodi operacija registriranja aplikacije. Za registriranje aplikacije, potrebni parametri su javni ključ aplikacije i model vlasnika aplikacije. Javni ključ aplikacije se dohvaća iz URL-a zahtjeva, a model vlasnika aplikacije iz tijela zahtjeva.



Sl. 4.55. Definicija REST zahtjeva za registriranje aplikacije

Rezultat izvođenja operacije registriranje aplikacije je REST odgovor, čija je definicija prikazana na slici 4.56. Model registrirane aplikacije je smješten u tijelo REST odgovora, a predstavljen kao JSON objekt.

Response Information

Resource Description

Application's info model.

ApplicationVM

Name	Description	Type	Additional information
AppKey	Application's public key.	string	Required
PrivateKey	Application's private key.	string	Required
DateCreated	Date created.	date	Required
OwnerId	Owner's identifier.	globally unique identifier	Required
ApiRootURL	Api root url.	string	Required

Response Formats

application/json, text/json

Sample:

```
{
  "AppKey": "sample string 1",
  "PrivateKey": "sample string 2",
  "DateCreated": "2018-06-27T18:23:08.07166+02:00",
  "OwnerId": "415d38c4-7f17-42cf-a486-b40ba610e018",
  "ApiRootURL": "sample string 5"
}
```

Sl. 4.56. Definicija REST odgovora nakon registriranja aplikacije

Promjena javnog ključa aplikacije je druga operacija unutar funkcionalnosti upravljanja aplikacijama. Na slici 4.57 prikazana je definicija zahtjeva kojim se izvodi operacija promjene javnog ključa aplikacije. Za promjenu javnog ključa aplikacije, potrebni parametri su trenutni javni ključ aplikacije, privatni ključ aplikacije i novi javni ključ. Trenutni javni ključ i privatni ključ aplikacije se dohvaćaju iz URL-a zahtjeva, a novi javni ključ aplikacije iz tijela zahtjeva.

PUT apps/{appKey}/changeKey/{privateAppKey}

PUT route for changing application public key.

Request Information

URI Parameters

Name	Description	Type	Additional information
appKey	Application public key.	string	Required
privateAppKey	Application's private key.	string	Required

Body Parameters

New application public key.

string

Request Formats

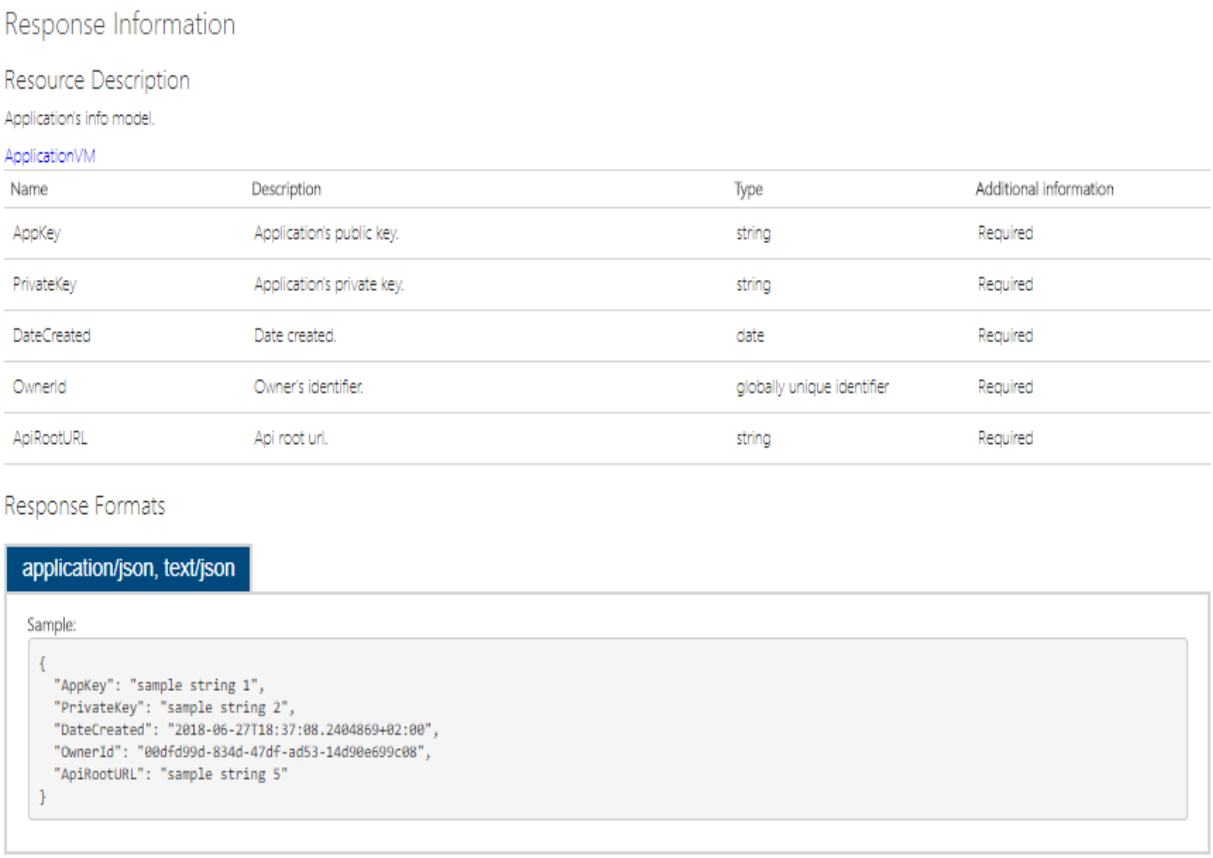
application/json, text/json

Sample:

```
"sample string 1"
```

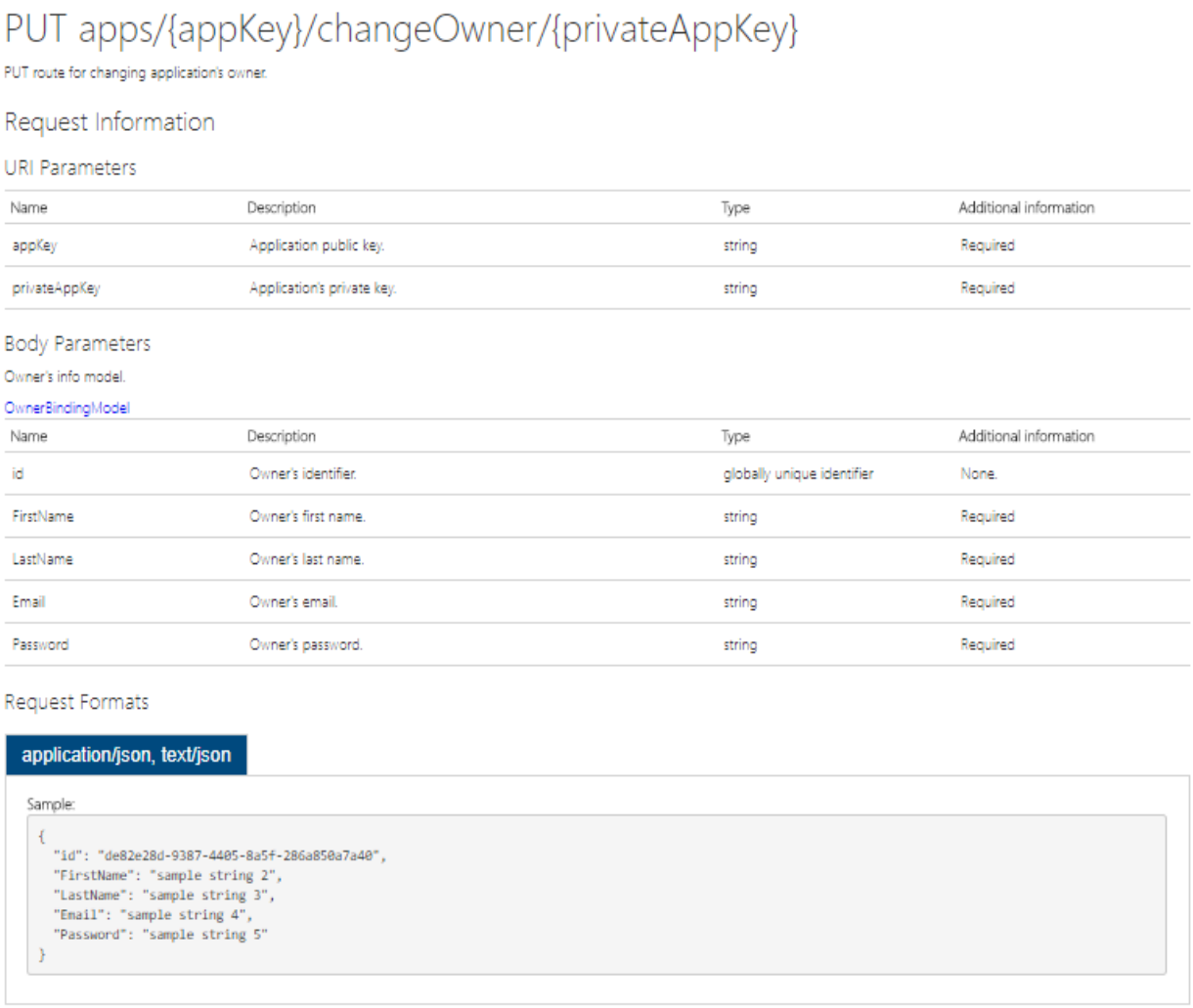
Sl. 4.57. Definicija REST zahtjeva za promjenu javnog ključa aplikacije

Rezultat izvođenja operacije promjene javnog ključa aplikacije je REST odgovor, čija je definicija prikazana na slici 4.58. Ažurirani model aplikacije je smješten u tijelo REST odgovora, a predstavljen kao JSON objekt.



Sl. 4.58. Definicija REST odgovora nakon promjene javnog ključa aplikacije

Promjena vlasnika aplikacije je treća operacija unutar funkcionalnosti upravljanja aplikacijama. Na slici 4.59 prikazana je definicija zahtjeva kojim se izvodi operacija promjene vlasnika aplikacije. Za promjenu vlasnika aplikacije, potrebni parametri su javni ključ aplikacije, privatni ključ aplikacije i model novog vlasnika aplikacije. Javni ključ i privatni ključ aplikacije se dohvaćaju iz URL-a zahtjeva, a model novog vlasnika iz tijela zahtjeva.



PUT apps/{appKey}/changeOwner/{privateAppKey}

PUT route for changing application's owner.

Request Information

URI Parameters

Name	Description	Type	Additional information
appKey	Application public key.	string	Required
privateAppKey	Application's private key.	string	Required

Body Parameters

Owner's info model.

OwnerBindingModel

Name	Description	Type	Additional information
id	Owner's identifier.	globally unique identifier	None.
FirstName	Owner's first name.	string	Required
LastName	Owner's last name.	string	Required
Email	Owner's email.	string	Required
Password	Owner's password.	string	Required

Request Formats

application/json, text/json

Sample:

```
{
  "id": "de82e28d-9387-4405-8a5f-286a850a7a40",
  "FirstName": "sample string 2",
  "LastName": "sample string 3",
  "Email": "sample string 4",
  "Password": "sample string 5"
}
```

Sl. 4.59. Definicija REST zahtjeva za promjenu vlasnika aplikacije

Rezultat izvođenja operacije promjene vlasnika aplikacije je REST odgovor, čija je definicija prikazana na slici 4.60. Ažurirani model aplikacije je smješten u tijelo REST odgovora, a predstavljen kao JSON objekt.

Response Information

Resource Description

Application's info model.

ApplicationVM

Name	Description	Type	Additional information
AppKey	Application's public key.	string	Required
PrivateKey	Application's private key.	string	Required
DateCreated	Date created.	date	Required
OwnerId	Owner's identifier.	globally unique identifier	Required
ApiRootURL	Api root url.	string	Required

Response Formats

application/json, text/json

Sample:

```
{
  "AppKey": "sample string 1",
  "PrivateKey": "sample string 2",
  "DateCreated": "2018-06-27T18:50:40.5786355+02:00",
  "OwnerId": "18e3fef6-64b6-422e-9630-10c2cf44fc48",
  "ApiRootURL": "sample string 5"
}
```

Sl. 4.60. Definicija REST odgovora nakon promjene vlasnika aplikacije

Dohvaćanje informacija o aplikaciji je četvrta operacija unutar funkcionalnosti upravljanja aplikacijama. Na slici 4.61 prikazana je definicija zahtjeva kojim se izvodi operacija dohvaćanja informacija o aplikaciji. Za dohvaćanje informacija, potrebni parametri su javni ključ aplikacije i privatni ključ aplikacije, koji se dohvaćaju iz URL-a zahtjeva.

GET apps/{appKey}/{privateAppKey}

GET route for getting application's info model.

Request Information

URI Parameters

Name	Description	Type	Additional information
appKey	Application public key.	string	Required
privateAppKey	Application's private key.	string	Required

Body Parameters

None.

Sl. 4.61. Definicija REST zahtjeva za dohvaćanje informacija o aplikaciji

Rezultat izvođenja operacije dohvaćanja informacija o aplikaciji je REST odgovor, čija je definicija prikazana na slici 4.62. Model aplikacije je smješten u tijelo REST odgovora, a predstavljen kao JSON objekt.

Response Information

Resource Description

Application's info model.

ApplicationVM

Name	Description	Type	Additional information
AppKey	Application's public key.	string	Required
PrivateKey	Application's private key.	string	Required
DateCreated	Date created.	date	Required
OwnerId	Owner's identifier.	globally unique identifier	Required
ApiRootURL	Api root url.	string	Required

Response Formats

application/json, text/json

Sample:

```
{
  "AppKey": "sample string 1",
  "PrivateKey": "sample string 2",
  "DateCreated": "2018-06-27T19:00:43.5447738+02:00",
  "OwnerId": "d9bd5725-cf04-4ec0-abc8-fe77801442ef",
  "ApiRootURL": "sample string 5"
}
```

Sl. 4.62. Definicija REST odgovora nakon dohvaćanja informacija o aplikaciji

Brisanje aplikacije je peta operacija unutar funkcionalnosti upravljanja aplikacijama. Na slici 4.63 prikazana je definicija zahtjeva kojim se izvodi operacija brisanja aplikacije. Za brisanje aplikacije, potrebni parametri su javni ključ aplikacije i privatni ključ aplikacije, koji se dohvaćaju iz URL-a zahtjeva.

DELETE apps/{appKey}/{privateAppKey}

DELETE route for deleting an application.

Request Information

URI Parameters

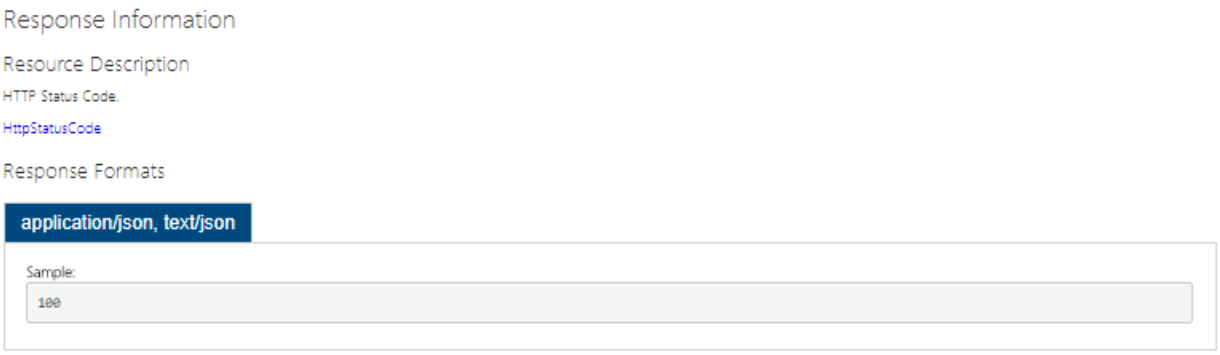
Name	Description	Type	Additional information
appKey	Application public key.	string	Required
privateAppKey	Application's private key.	string	Required

Body Parameters

None.

Sl. 4.63. Definicija REST zahtjeva za brisanje aplikacije

Rezultat izvođenja operacije brisanja aplikacije je REST odgovor, čija je definicija prikazana na slici 4.64.

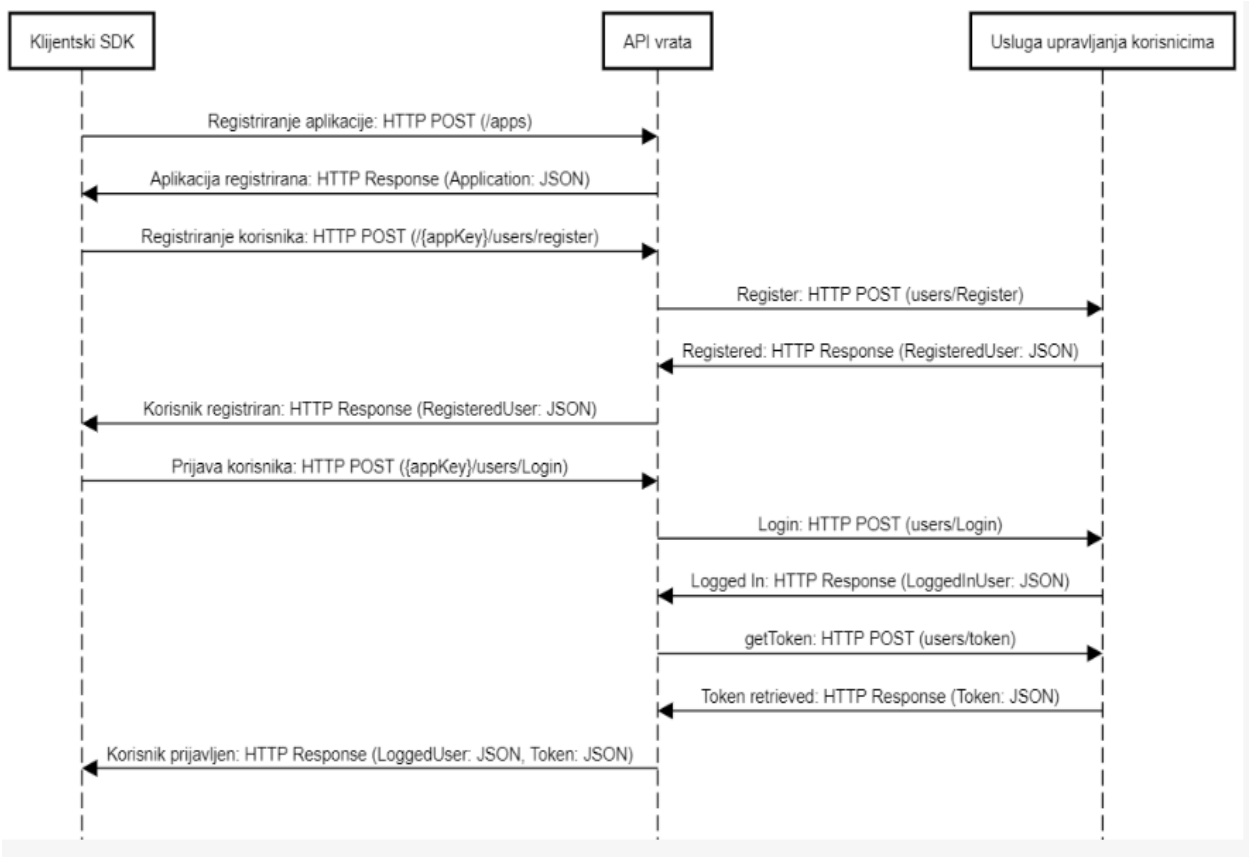


Sl. 4.64. Definicija REST odgovora nakon brisanja aplikacije

4.5.2. Filtriranje i raspoređivanje prometa

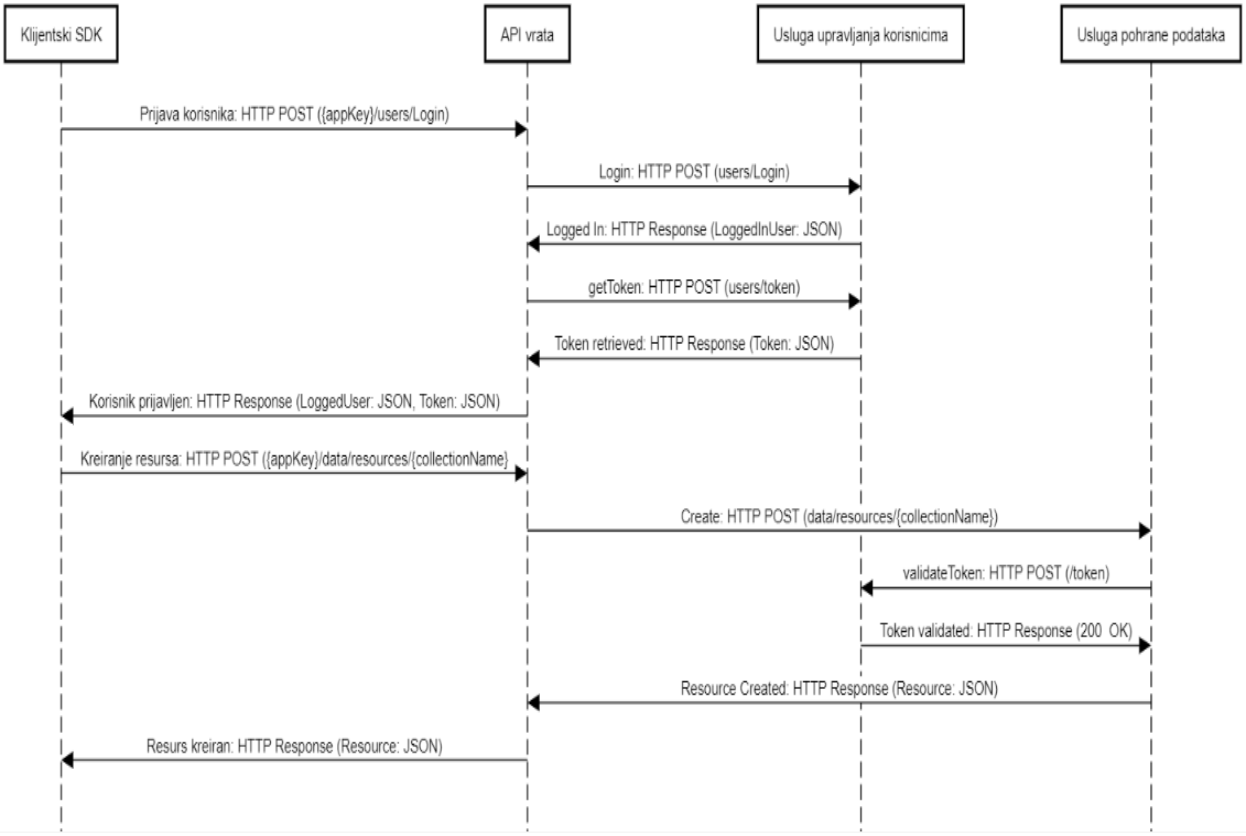
Jedna od funkcionalnosti API vrata sustava BaaS je između ostalog i filtriranje i raspoređivanje prometa. U sustavu BaaS, razvijenom u ovom radu, pod prometom se podrazumijevaju HTTP zahtjevi i HTTP odgovori. Zbog činjenice da su API vrata ulazna točka za sve klijentske zahtjeve, ostvaruje se mogućnost skrivanja interne strukture sustava BaaS. Klijent ne mora znati točnu strukturu REST zahtjeva kojim se izvode operacije neke usluge. API vrata sustava BaaS definiraju vlastite, predstavljajuće strukture zahtjeva, za izvođenjem operacija usluga sustava BaaS. Iz svakog takvog zahtjeva se izdvaja javni ključ aplikacije, te se pozivaju operacije usluga sustava BaaS, određenim redoslijedom i prateći internu strukturu usluga.

Većina operacija usluga sustava BaaS je dostupna samo ovlaštenim korisnicima. Korisnici su ovlašteni za izvođenje neke operacije, ako su registrirani unutar klijentske aplikacije i pripadaju odgovarajućoj ulozi. Za registriranje korisnika i ovjeru autentičnosti, zadužena je usluga upravljanja korisnicima. Usluga upravljanja korisnicima koristi protokol OAuth2 za prijavu i ovjeru autentičnosti korisnika. Svaki korisnik nakon prijave dobije vlastitu pristupnu značku koju tada mora smjestiti u polje *Authorization* svakog idućeg zahtjeva. Na slici 4.65 prikazan je slijedni dijagram koji prikazuje scenarij registriranja i prijave korisnika.



Sl. 4.65. Sekvencijalni dijagram registriranja i prijave korisnika

Nakon prijave, korisnik stječe pristupnu značku kojom postaje ovlašten za izvođenje operacija ostalih usluga sustava BaaS. API vrata sustava BaaS na ovaj način vrše funkcionalnost filtriranja i raspoređivanja prometa, kako bi se ispunio zahtjev klijenta. No, API vrata ne služe kao posrednik u komunikaciji među uslugama. Na taj način bi se zakomplicirala komunikacija, a nepotrebni zahtjevi bi zagušili API vrata i usporile rad sustava BaaS. Osim toga, time se sustav približava monolitnoj arhitekturi a udaljava od arhitekture mikrousluga. Komunikacije međusobno komuniciraju REST zahtjevima, s obzirom da su po strukturi sve usluge razvijene kroz okvir web API i temelje se na REST arhitekturi. Na slici 4.66 prikazan je sekvencijalni dijagram koji prikazuje scenarij kreiranja resursa kroz uslugu pohrane podataka.



Sl. 4.66. Sekvencijalni dijagram kreiranja resursa

5. UPOTREBA SUSTAVA BaaS U RAZVOJU WEB APLIKACIJE

U ovom poglavlju, prikazana je upotreba razvijenog sustava BaaS u razvoju klijentske web aplikacije. Kao posrednik između sustava BaaS i web aplikacije, razvijen je paket za razvoj programa (engl. *SDK - Software Development Kit*) u tehnologiji kojom se razvija i web aplikacija. Razvijeni SDK olakšava povezivanje web aplikacije s pozadinskim uslugama sustava BaaS te sakriva njegovu internu strukturu od krajnjeg korisnika. Izrada same web aplikacije na ovaj način, trebala bi demonstrirati glavne prednosti sustava BaaS, a to su povećana brzina i jednostavnost razvoja.

5.1. Razvoj klijentskog SDK-a

5.1.1. Razvojni okvir Angular

Angular je razvojni okvir koji olakšava izradu web aplikacija. Kombinira deklarativne predloške, ubrizgavanje ovisnosti i ostale integrirane najbolje prakse za rješavanje razvojnih izazova [37]. Omogućuje programerima izradu aplikacija različitog tipa, kao što su aplikacije za web, mobilne uređaje ili radne površine. Angular je razvojni okvir otvorenog koda, a temelji se na programskom jeziku TypeScript. TypeScript je programski jezik otvorenog koda kojeg je razvio i održava Microsoft. To je strog sintaktični nadskup programskog jezika JavaScript, a dodaje neobavezno statičko upisivanje. Dizajniran je za razvoj velikih aplikacija, a prevodi se u JavaScript kod. TypeScript se može koristiti za razvoj JavaScript aplikacija za izvršenje na strani klijenta i na strani poslužitelja.

Razvojni okvir Angular implementira osnovnu i neobaveznu funkcionalnost kao skup biblioteka pisanih u programskom jeziku TypeScript, koje se uvode u Angular aplikaciju. Osnovni građevni blokovi Angular aplikacije se nazivaju *NgModules*, a omogućavaju sastavljanje konteksta za komponente. Prikupljaju povezane kodove u funkcionalne skupove i Angular aplikacija je definirana skupom takvih modula. Angular aplikacija uvijek ima korijenski modul koji omogućuje početno pokretanje (engl. *bootstrapping*) i sastavljanje ostalih modula. Komponente u razvojnom okviru Angular definiraju poglede, koji su skup elemenata zaslona koje okvir može odabrati i mijenjati prema programskoj logici i podacima. Komponente koriste usluge, koje pružaju određenu funkcionalnost koja nije izravno povezana s pogledima. Davatelji usluga (engl. *service providers*) mogu se ubaciti u komponente kao ovisnosti, što čini programski kod modularnim, ponovljivim i učinkovitim.

5.1.2. Upravitelj paketa npm

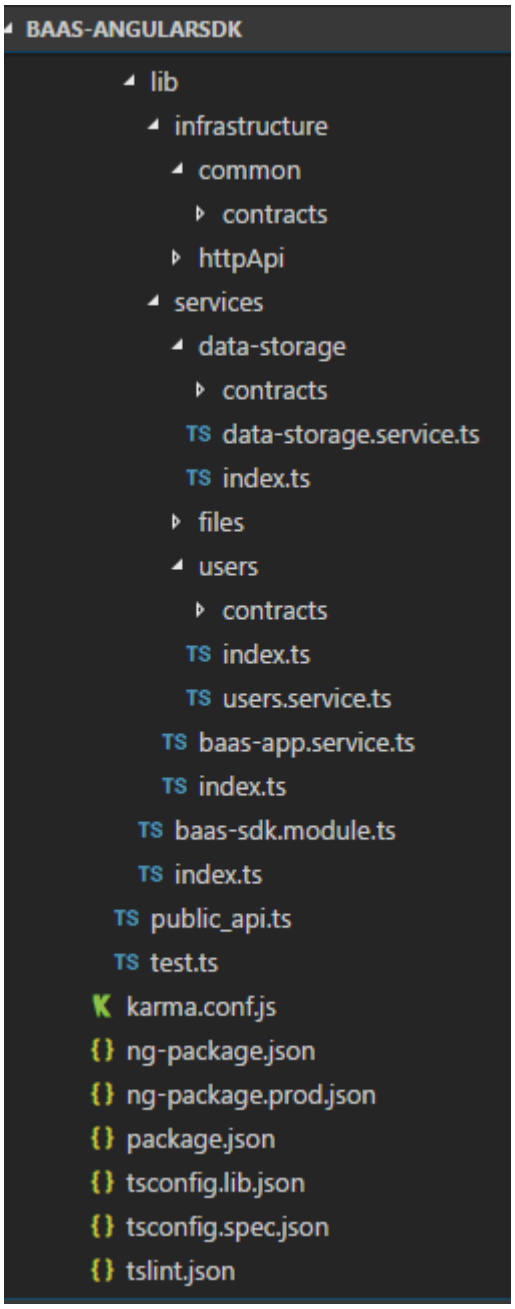
Npm je upravitelj paketa za programski jezik JavaScript te standardni upravitelj paketa za platformu Node.js [38]. Ujedno je i najveći registar programa na svijetu, s oko 3 milijarde preuzimanja tjedno. Npm se sastoji od tri dijela:

- Web stranice - služi za pretraživanje paketa, postavljanje profila i upravljanje drugim aspektima upravitelja paketa npm.
- Sučelja komandne linije - pokreće se iz terminala i omogućuje interakciju s programerima.
- Registar - velika javna baza podataka JavaScript programa i pripadnih meta-podataka.

Npm može upravljati paketima koji su u obliku lokalnih ovisnosti pojedinog projekta, ali i u obliku globalno instaliranih JavaScript alata. Kad se koristi kao upravitelj ovisnosti za lokalni projekt, npm može instalirati sve pakete koji su definirani kao ovisnosti projekta u datoteci *package.json*. U datoteci *package.json* je za svaku ovisnost definiran niz valjanih verzija, pomoću sheme semantičke verzije, omogućujući razvojnim programerima da automatski ažuriraju svoje pakete, dok istodobno izbjegavaju neželjene promjene. Npm također pruža datoteku *package-lock.json*, koja sadrži unos točne inačice koju koristi projekt nakon procjene semantičke verzije u datoteci *package.json*.

5.1.3. Struktura razvijenog SDK-a

Razvijeni SDK pruža integracijski pristup svim operacijama razvijenog sustava BaaS. Razvijen je kao biblioteka u razvojnom okviru Angular, koja pruža usluge za upotrebu pri razvoju klijentske web aplikacije. Pojedinačna usluga biblioteke odgovara jednoj usluzi sustava BaaS te sadrži programski kod za uspostavljanje komunikacije s pripadajućom uslugom sustava BaaS. Biblioteka se jednostavno kreira koristeći ugrađene mehanizme u komandnom sučelju za razvojni okvir Angular. Sadržaj programskog rješenja razvijenog SDK-a je prikazan na slici 5.1. Korijenski modul biblioteke je modul naziva *BaasSdkModule*, te se u njemu vrši unos svih ostalih usluga. Usluge su grupirane u direktoriju *services*, a u direktoriju *infrastructure* nalaze se zajedničke datoteke, sučelja i ugovori koji su potrebni za rad usluge. U svaku uslugu se ubrizgava ovisnost o uslugama *HttpClient* i *BaasService*. Na slici 5.2 prikazan je programski kod za uslugu *BaasService*. Pomoću te usluge se dohvaćaju inicijalizacijski podaci koje klijent konfigurira prilikom instalacije razvijenog SDK-a u web aplikaciju. Osnovni inicijalizacijski podatak je javni ključ aplikacije, koji jednoznačno identificira aplikaciju. Usluga *HttpClient* je dio jedne od ugrađenih biblioteka za razvojni okvir Angular, a sadrži funkcije za izvođenje HTTP operacija.



Sl. 5.1. Sadržaj programskog rješenja razvijenog SDK-a

```
import { Injectable, Inject } from '@angular/core';

import { CONFIGURATION } from '../infrastructure/common/contracts';

@Injectable()
export class BaasService {
  apiKey: string;

  constructor( @Inject('config') config: CONFIGURATION) {
    this.apiKey = config.ApiKey;
  }
};
```

Sl. 5.2. Programski kod za dohvaćanje konfiguracijskih podataka klijentske aplikacije

Većina operacija usluga razvijenog SDK-a zahtijeva pristupnu značku za autorizaciju HTTP zahtjeva. Pristupna značka se dohvaća iz lokalne pohrane Internet preglednika u kojem se izvršava web aplikacija. Razvijeni Angular SDK automatski manipulira pristupnom značkom na način da ju pohranjuje i uklanja iz lokalne pohrane kroz operacije usluge upravljanja korisnicima. Na slici 5.3 prikazan je programski kod za izvođenje zahtjeva za operacijom prijave korisnika. Kao odgovor se očekuje pristupna značka koji služi za ovjeru autentičnosti korisnika aplikacije.

```
login(data: ILoginUser): Observable<HttpResponse<IAppUser>> {
        let httpOptions = {
          observe: 'response',
          headers: new HttpHeaders({
            'Content-Type': 'application/x-www-form-urlencoded',
            'Accept': 'application/json'
          }),
          params: new HttpParams().set('username',
data.userName).append('password', data.password).append('grant_type', 'password')
        };
        return this.http.post(UsersEndpoints.token,
httpOptions).pipe(switchMap(response => {
          if (response['_body']['access_token']) {
            localStorage.setItem('access_token',
JSON.stringify(response['_body']['access_token']));
            let loginHttpOptions = {
              observe: 'response',
              headers: new HttpHeaders({
                'Content-Type': 'application/json',
                'Accept': 'application/json'
              }),
            };
            return this.http.post(UsersEndpoints.login, data, loginHttpOptions);
          }
        })
        );
      }
```

Sl. 5.3. Programski kod za izvođenje zahtjeva za operacijom prijave korisnika

Sve operacije koje su u sastavu razvijenog SDK-a, pisane su u skladu s operacijama usluga sustava BaaS. S obzirom da su usluge sustava BaaS razvijene kao web API-ji, u operacijama razvijenog SDK-a se formiraju HTTP zahtjevi i obrađuju HTTP odgovori. Sve operacije su asinkrone, što znači da ne blokiraju rad klijentske aplikacije. Na slici 5.4 prikazan je programski kod za izvođenje zahtjeva za operacijom kreiranja resursa. Sve operacije vraćaju tip podataka *Observable* iz biblioteke *RxJS*, koji služi za pohranu asinkronih funkcija. Pohranjena funkcija se izvodi emitiranjem *Observable* varijable, ali tek kada se klijent pretplati na nju. U ovom slučaju, asinkrone funkcije su HTTP zahtjevi, a mogu emitirati dvije vrijednosti: HTTP odgovor ili pogrešku.

```
create(collectionName: string, data: string): Observable<HttpResponse<IResource>>
{
        let httpOptions = {
          observe: 'response',
          headers: new HttpHeaders({
            'Content-Type': 'application/json',
            'Accept': 'application/json',
            'Authorization': 'Bearer ' + localStorage.getItem('access_token')
          })
        };
        return this.http.post(DataStorageEndpoints.resources + collectionName,
data, { observe: 'response' });
}
```

Sl. 5.4. Programski kod za izvođenje zahtjeva za operacijom kreiranja resursa

Razvijeni SDK je moguće koristiti pri razvoju klijentske web aplikacije, pod uvjetom da se ona razvija u razvojnom okviru Angular. U tom slučaju, cjelokupni SDK se uključuje u aplikaciju kao vanjska biblioteka. S obzirom da su i osnovne biblioteke za razvojni okvir Angular dostupne preko upravitelja paketa npm, kao dobra praksa se nameće korištenje razvijenim SDK-om kao npm paketom. Da bi razvijeni SDK bio dostupan kao npm paket, potrebno ga je objaviti u registar upravitelja paketa npm. Prije same objave (engl. *publish*), potrebno je pokrenuti skripte za izgradnju (engl. *build*) paketa. Pri objavi razvijenog SDK-a u registar upravitelja paketa npm, uzima se u obzir samo kompilirani modul koji se nalazi u direktoriju *dist*, koji je nastao izgradnjom npm paketa. Direktorij *dist* također mora sadržavati datoteku *package.json*, u kojoj se specificira izgrađeni npm paket. Naposljetku, objava razvijenog SDK-a, kao npm paketa, se ostvaruje izvođenjem naredbe: *npm publish*. Slika 5.5 prikazuje objavljeni paket u registru upravitelja paketa npm. Ukoliko se, nakon nekog vremena, ažurira npm paket, potrebno je ponovno pokrenuti skripte za izgradnju paketa, promijeniti broj verzije, ažurirati zapisnik promjena i ponovno objaviti paket u registar upravitelja paketa npm.

Sl. 5.5. Razvijeni SDK kao objavljeni paket u registru upravitelja paketa npm

5.2. Razvoj klijentske web aplikacije

Uobičajeni razvoj web aplikacije uključuje paralelni razvoj sučelja i pozadinske podrške aplikacije. Usklađivanje paralelnog razvoja tih dvaju dijelova u praksi često nailazi na prepreke, a jedna od njih je komunikacija putem REST API-ja. U ovom potpoglavlju namjera je prikazati razvoj web aplikacije bez pisanja programskog koda za pozadinsku podršku web aplikacije. Razvijeni sustav BaaS pruža svu potrebnu pozadinsku podršku web aplikaciji posredstvom razvijenog SDK-a.

5.2.1. Konfiguracija razvijenog SDK-a unutar web aplikacije

Razvoj web aplikacije koja je klijent sustava BaaS, podrazumijeva razvoj u razvojnom okviru Angular. Pravilno konfiguriranje aplikacije, za upotrebu razvijenog SDK-a, zahtijeva dodavanje ovisnosti o objavljenom npm paketu na jedan od dva načina:

- Eksplicitnim navođenjem ovisnosti u datoteci *package.json* - *„@mdudjak/baas-sdk“: „0.1.0“*.
- Izvođenjem npm naredbe u komandnom sučelju - *npm install @mdudjak/baas-sdk --save*.

Da bi se razvijeni SDK mogao koristiti kao jedna od biblioteka unutar web aplikacije, potrebno je inicijalizirati SDK u glavnom modulu aplikacije, na sljedeći način: *import { BaasSdkModule } from ' @mdudjak/baas-sdk '*.

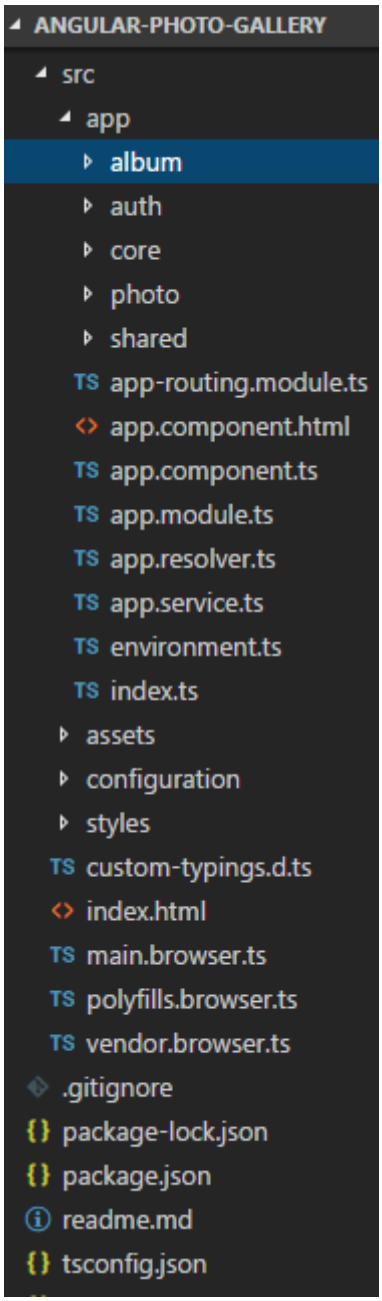
Razvijeni SDK u razvojnom okviru Angular, dozvoljava konfiguriranje postavki web aplikacije prilikom definiranja unosa u glavnom modulu aplikacije. Od postavki se navodi jedinstveni ključ aplikacije koji služi za identificiranje aplikacije prilikom slanja zahtjeva prema sustavu BaaS. Programski kod za definiranje postavki aplikacije je prikazan na slici 5.6. Za glavni modul objavljenog SDK-a je definiran unos unutar glavnog modula web aplikacije i označen dekoratorom *NgModule*, koji prikuplja meta-podatke o modulima i organizira povezane stvari zajedno.

```
@NgModule({
  ...,
  imports: [
    ...,
    baasSdkModule.forRoot({
            ApiKey: "api-key"
    })
  ],
  ...
})
```

Sl. 5.6. Programski kod za definiranje postavki aplikacije

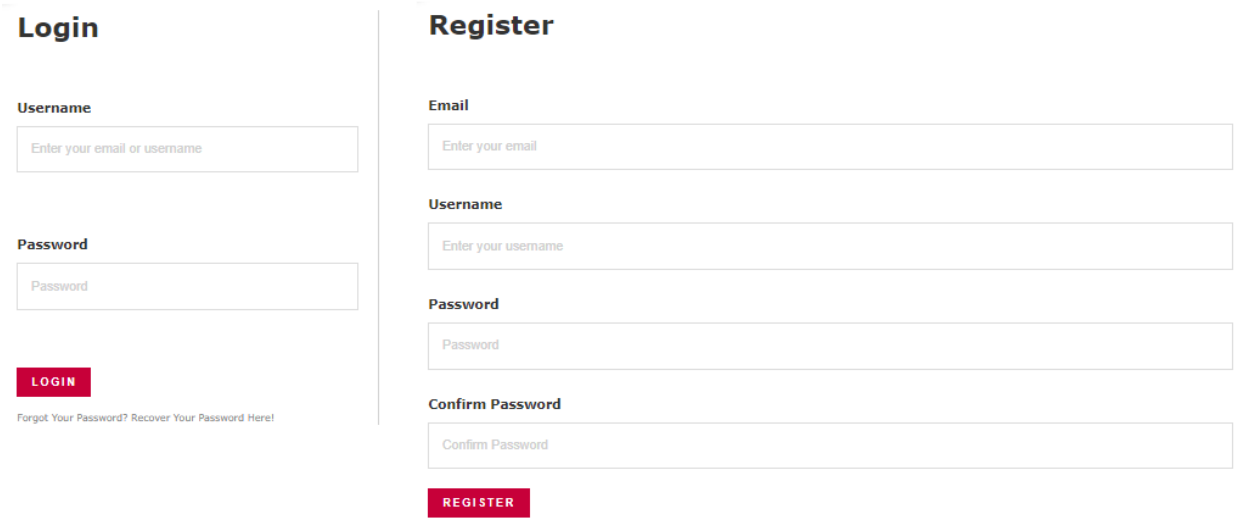
5.2.2. Razvoj web fotogalerije kroz razvojni okvir Angular

U svrhu prikaza lakoće razvoja web aplikacije koristeći funkcionalnosti sustava BaaS, prikazan je razvoj web fotogalerije pomoću razvijenog SDK-a u razvojnom okviru Angular. Web fotogalerija prikazuje fotografije i albume različitih korisnika, što ju čini vrlo dobrim primjerom web aplikacije koja koristi sve usluge razvijenog sustava BaaS. Dozvoljava registraciju više korisnika, a svaki korisnik ima mogućnosti objavljivanja različitih foto albuma. Time aplikacija koristi sve tri usluge sustava BaaS: uslugu upravljanja korisnicima, uslugu pohrane podataka i uslugu upravljanja datotekama. Sadržaj programskog rješenja razvijene web fotogalerije je prikazan na slici 5.7.



Sl. 5.7. Sadržaj programskog rješenja razvijene web fotogalerije

Razvijena web fotogalerija koristi neke operacije unutar usluge upravljanja korisnicima, sustava BaaS, kao što su registracija, prijava i odjava korisnika te ažuriranje lozinke. Sučelje web fotogalerije za prijavu i registraciju korisnika je prikazano na slici 5.8.



Sl. 5.8. Sučelje web fotogalerije za prijavu i odjavu korisnika

Uobičajeni razvoj web aplikacija u razvojnom okviru Angular podrazumijeva definiranje vlastitih komponenti, koje su osnovni građevni blokovi korisničkog sučelja aplikacije. Svaka komponenta u razvijenoj web fotogaleriji sadrži predložak za opisni kod sučelja u opisnim jezicima HTML i CSS, a logika za interakciju s korisnikom i obradu podataka je pisana u programskom jeziku TypeScript. Da bi se usluge sustava BaaS koristile u komponentama web fotogalerije, potrebno ih je ubrizgati kao ovisnost u te komponente, kako je prikazano na slici 5.9. Potom se usluge sustava BaaS predstavljaju kao usluge unutar komponenata sučelja web fotogalerije, a podnošenje zahtjeva za operacijama sustava BaaS se jednostavno izvodi pozivanjem funkcija ubrizganih usluga. Slika 5.9 također prikazuje i programski kod odgovoran za registriranje i prijavu korisnika unutar razvijene web fotogalerije.

```
…
import { UsersService } from @mdudjak/baas-sdk;
…

async register(): Promise<void> {
        let user = this.form.getRawValue();
        user.creationDate = new Date();
        this.usersService.account.register(user).subscribe((response => {
            this.message = 'You have successfully registered!';
            this.router.navigate(['/login']);
        }),
            error => {
                this.message = error.statusCode + ': ' + error.data.message;
            });
    }

    async login(): Promise<void> {
        let user = this.form.getRawValue();
        this.usersService.account.login(user).subscribe((response => {
            let user = response["body"];
            this.router.navigate([currentPath !== '/login' ? currentPath :
'/profile/' + user.id]);
        }),
            error => {
                this.message = error.statusCode + ': ' + error.data.message;
            });
    }
```

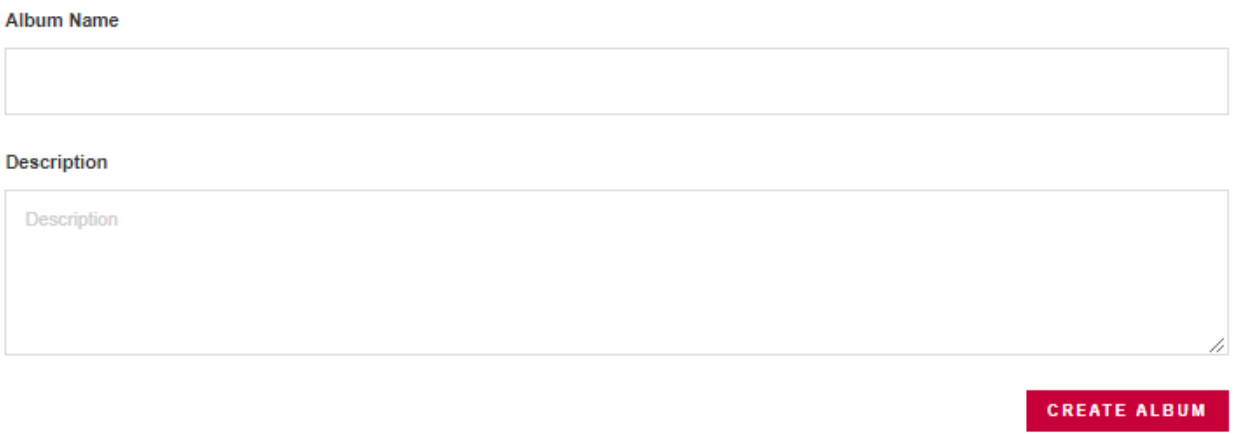
Sl. 5.9. Programski kod za registraciju i prijavu korisnika

Svaki korisnik unutar web fotogalerije ima mogućnost objave više foto albuma. U kontekstu web fotogalerije, foto album je podatkovni objekt koji ima odgovarajuće atribute koji ga opisuju. Svaki foto album sadrži jednu ili više fotografija, pri čemu je fotografija drugi podatkovni objekt unutar web fotogalerije. Za modeliranje baze podataka web fotogalerije, koristi se usluga pohrane podataka sustava BaaS. S obzirom na nerelacijsku bazu podataka usluge, definirane su dvije različite sheme zbirki resursa: album i fotografija. Strukture shema su prikazane na slici 5.10. U zbirku albuma se svrstavaju resursi koji sadrže opisne podatke o albumima, a sadrže i listu identifikatora pripadnih resursa zbirke fotografija. U zbirku fotografija se svrstavaju resursi koji sadrže opisne podatke o fotografijama, a sadrže i URL za preuzimanje prenesenih fotografija.

```
export class Album {
     name:string;
     description:string;
     coverId:string;
     dateCreated:Date;
     dateUpdated:Date;
     userId:string;
     photoID:string[];
}

export class Photo {
    title:string;
    description:string;
    dateCreated:Date;
    photoURL:string;
}
```

Sl. 5.10. Shema zbirki resursa albuma i fotografija

Web fotogalerija sadrži sučelja za kreiranje albuma te sučelja za prijenos fotografija koje pripadaju određenom albumu. Nakon uspješne registracije i prijave, korisnik je preusmjeren na početno sučelje koje prikazuje sve njegove albume. Ukoliko korisnik ne posjeduje niti jedan album, preusmjeren je na sučelje za kreiranje albuma, koje je prikazano na slici 5.11.



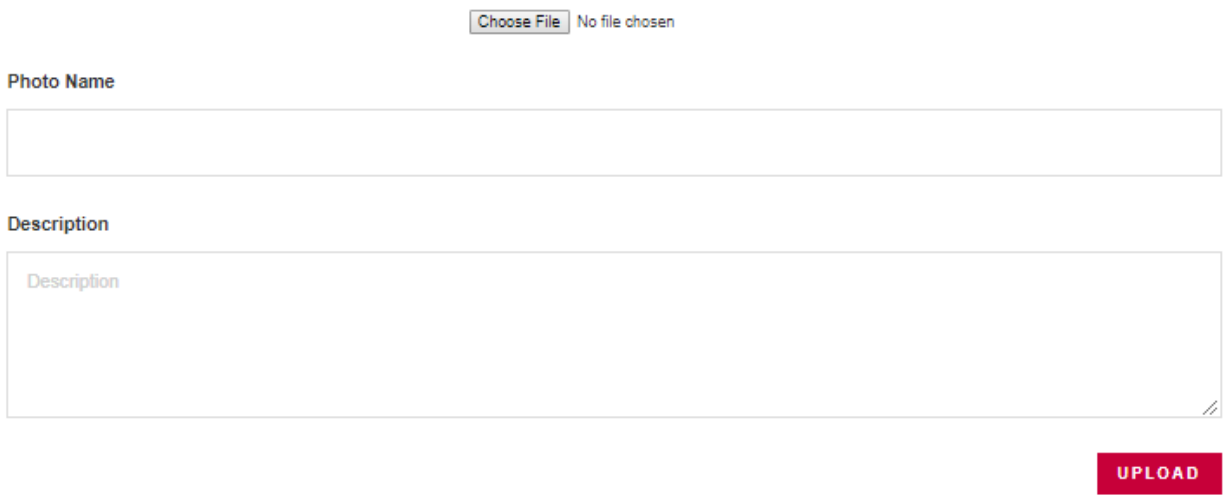
Sl. 5.11. Sučelje web fotogalerije za kreiranje albuma

Sve CRUD operacije za rad sa zbirkama albuma i fotografija, te pojedinačnim resursima, pružene su u sastavu usluge pohrane podataka sustava BaaS. Kao i kod usluge upravljanja korisnicima, potrebno je ubrizgati ovisnost o toj usluzi unutar komponente sučelja web fotogalerije i pozivati njoj pripadne funkcije. Programski kod za kreiranje albuma je prikazan na slici 5.12.

```
…
import { UsersService, DataStorageService } from '@mdudjak/baas-sdk';
…

export class AlbumCreateRoute implements OnInit {
    user:any;
    album:Album;
    constructor(private dataStorageService: DataStorageService, private router:
Router,private usersService: UsersService){}

    async ngOnInit() : Promise<void>{
        this.user = await this.usersService.user.getUser();
    }

async createAlbum(): Promise<void> {
        var name: string = this.form.controls.albumName.value;
        var description: string = this.form.controls.albumDescription.value;
        this.album = new Album(name, description);
        var date = new Date();
        album.dateUpdated = album.dateCreated = date.toUTCString();
        album.userId = this.user.id;
        await this.dataStorageService.resource.create("Album", album);
        let url = '/profile/' + this.user.id + "/";
        this.router.navigate([url]);
    }
```

Sl. 5.12. Programski kod za kreiranje albuma

Nakon kreiranja albuma, korisnik ima mogućnost prijenosa fotografija koje se vezuju uz kreirani album. Sučelje za prijenos fotografije je prikazano na slici 5.13. Ono se pojavljuje automatski nakon završetka kreiranja albuma, tako da se prenesene fotografije odmah svrstavaju u kreirani album.



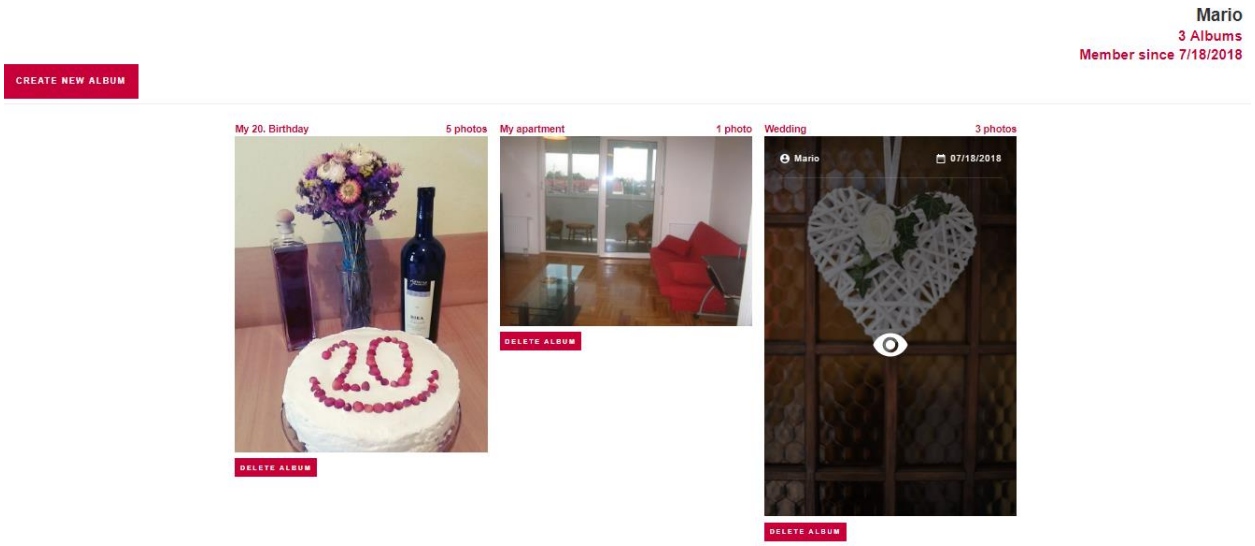
Sl. 5.13. Sučelje web fotogalerije za prijenos fotografije

Za prijenos fotografije u web fotogaleriju, koriste se operacije usluge upravljanja datotekama sustava BaaS. Kao ostale usluge sustava BaaS, uslugu upravljanja datotekama je potrebno ubrizgati kao ovisnost u odgovarajuću komponentu sučelja web fotogalerije. Programski kod za prijenos fotografije i svrstavanje u pripadni album se nalazi na slici 5.14. Koristi se operacija prijenosa datoteke usluge upravljanja datotekama i operacija kreiranja resursa usluge pohrane podataka. Stvara se resurs unutar kolekcije fotografija, a resursu se pridodaje atribut koji predstavlja URL na kojem se može preuzeti prenesena fotografija. Također se poziva i operacija ažuriranja resursa, kako bi se ažurirao resurs koji predstavlja album u koji se dodaje fotografija. U listu identifikatora fotografija resursa albuma se dodaje i identifikator resursa koji opisuje upravo prenesenu fotografiju.

```
…
Import { FilesService, DataStorageService } from '@mdudjak/baas-sdk';
…

export class PhotoUploadComponent implements OnInit {
    …
    constructor(private filesService: FilesService,
        private router: Router,
        private dataStorageService: DataStorageService) {}

    async uploadPhoto() {
        var title: string = this.form.controls.photoName.value + this.extension;
        var description: string = this.form.controls.photoDescription.value;
        let photo = new Photo(title, description);
        var date = new Date();
        photo.dateCreated = date.toUTCString();
        photo.photoURL = await this.filesService.create(this.file, title);
        photo = await this.dataStorageService.resource.create("Photo", photo);
        await this.insertPhoto(this.albumId, photo.id);
        let url = "/album/detail/" + this.albumId;
        this.router.navigate([url]);
    }

    async insertPhoto(albumId:string,photoId:string):Promise<void>{
        var album = await this.dataStorageService.resource.get("Album",albumId);
        var album_photos:string[] = album["photoID"];
        album_photos.push(photoId);
        album["photoID"]=album_photos;
        var date = new Date();
        album["dateUpdated"]= date.toUTCString();
        await this.dataStorageService.resource.update("Album",album);
    }
}
```

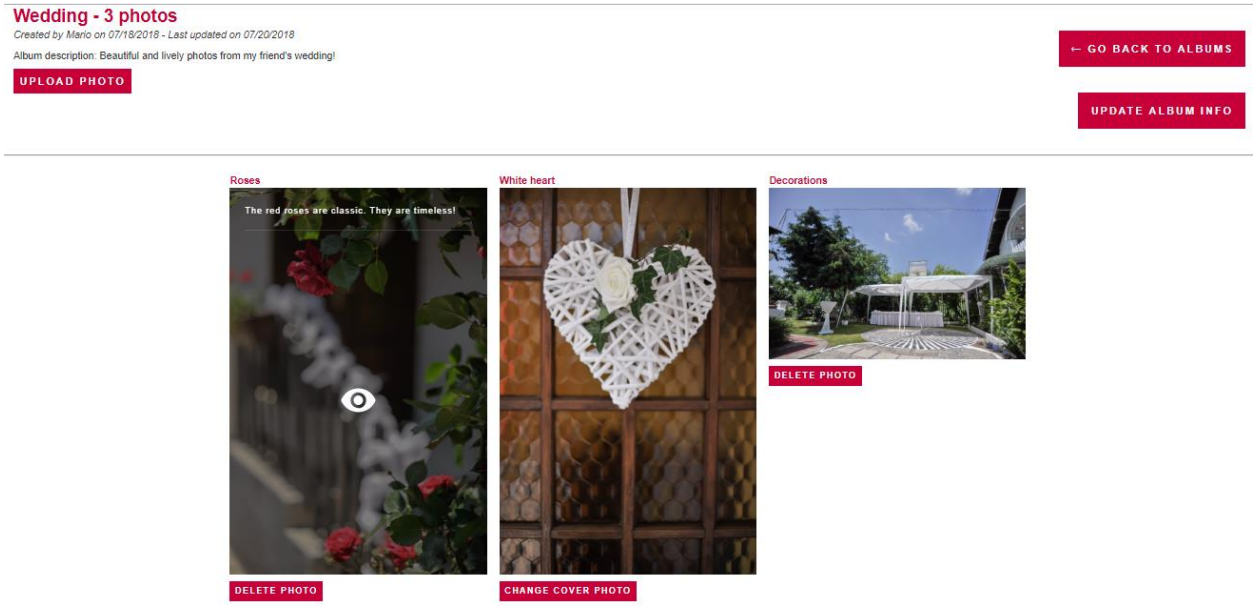
Sl. 5.14. Programski kod za prijenos fotografije i svrstavanje u pripadni album

Kada je korisnik kreirao album i obavio prijenos barem jedne fotografije unutar albuma, omogućen mu je pristup početnom sučelju, koje prikazuje listu njegovih albuma. Početno sučelje za prikaz liste albuma korisnika je prikazano na slici 5.15.



Sl. 5.15. Sučelje web fotogalerije za prikaz liste albuma korisnika

Svaki album je predstavljen svojom naslovnom slikom, a prijelazom preko slike se prikazuje pregled osnovnih informacija o albumu. Pritiskom na naslovnu sliku, prikazuje se sučelje za prikaz svih fotografija odabranog albuma, kako je prikazano na slici 5.16. Sučelje također pruža funkcionalnosti za prijenos novih fotografija u album, brisanje postojećih fotografija i ažuriranje informacija o albumu.



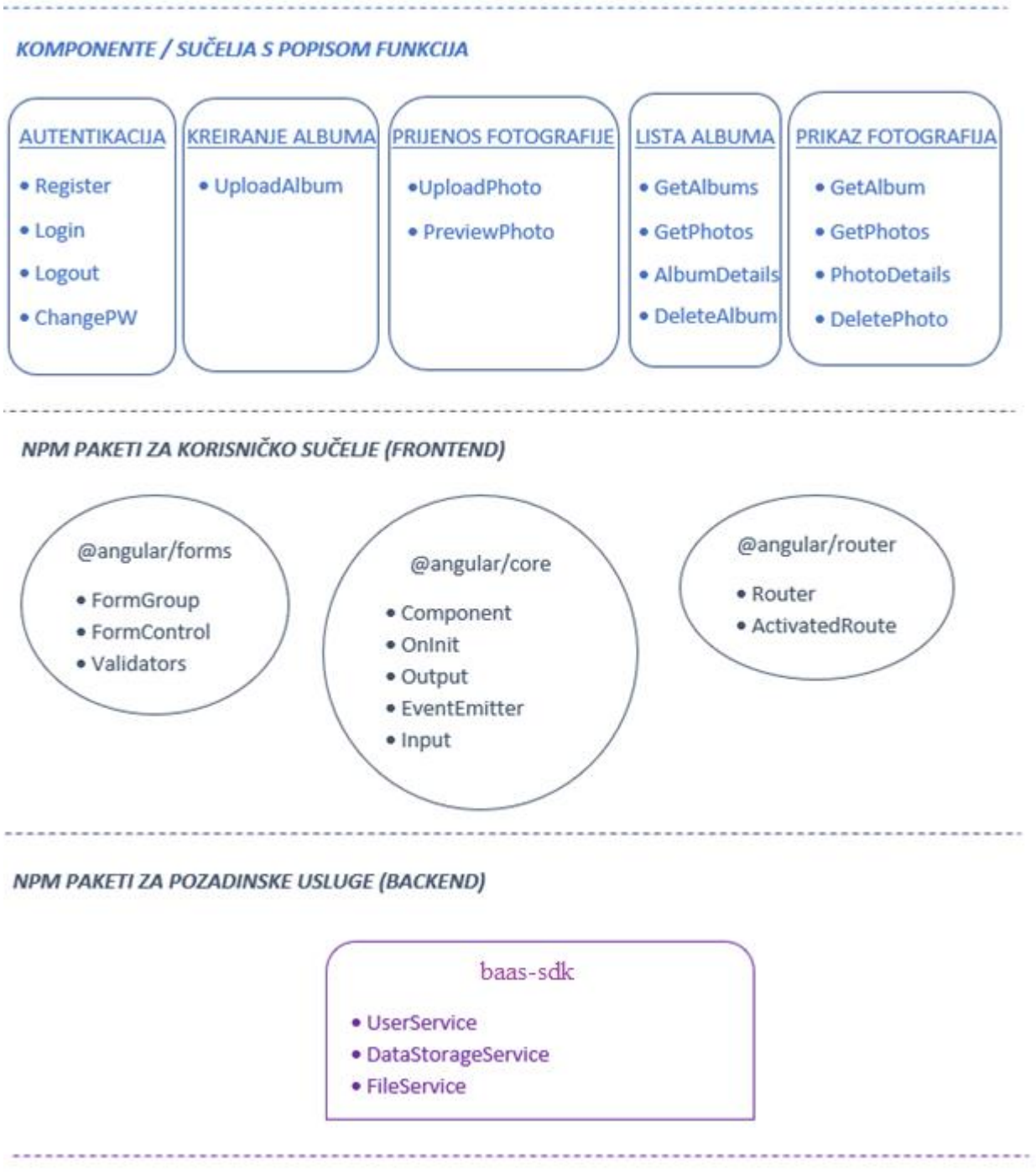
Sl. 5.16. Sučelje za prikaz fotografija u albumu

Operacije dohvaćanja albuma i pripadnih fotografija ostvaruju se HTTP GET zahtjevima prema usluzi pohrane podataka sustava BaaS. S druge strane, operacije ažuriranja i brisanja se ostvaruju HTTP PUT i HTTP DELETE zahtjevima prema istoj usluzi. Kao što je prikazano kod kreiranja albuma i prijenosa fotografije, ti zahtjevi se ostvaruju pozivanjem funkcija usluga razvijenog SDK-a. Kako bi usluge razvijenog SDK-a bile dostupne unutar klijentske web aplikacije, potrebno je za njih definirati unos na vrhu komponente sučelja i ubrizgati ovisnost o tim uslugama.

5.3. Analiza klijentske web aplikacije

Razvijena web fotogalerija zadovoljava osnovne uvjete koji se nameću aplikaciji tog tipa, kao što su: mogućnost registracije korisnika, kreiranje i razlikovanje korisničkih albuma, prijenos fotografija, ažuriranje informacija o albumima i fotografijama i drugi. Za ispunjavanje tih uvjeta, web fotogalerija koristi uslugu pohrane podataka sustava BaaS za rad s albumima i fotografijama kao podatkovnim objektima aplikacije, uslugu upravljanja datotekama za prijenos i prikazivanje fotografija te uslugu upravljanja korisnicima za ovjeru autentičnosti i upravljanje korisničkim podacima. Programeri web fotogalerije se ne moraju nositi s postavljanjem, funkcioniranjem i održavanjem pozadinske podrške. Stoga mogu više vremena utrošiti na razvijanje korisničkog iskustva, funkcionalnosti i korisničkog sučelja.

Nakon razvoja web fotogalerije, koja koristi usluge sustava BaaS kao pozadinsku podršku, provedena je analiza aplikacije. Analiza razvijene web fotogalerije podrazumijeva razlaganje aplikacije na komponente, s namjerom da se odredi raspodjela posla na pozadinske usluge i usluge prednjeg sučelja. Cilj analize je utvrditi udio pozadinskih usluga u web fotogaleriji, kako bi se razjasnilo koliko se ubrzava razvoj web aplikacije i za koliko se smanjuje njena složenost. U prvom koraku analize, dizajniran je dijagram komponenti web fotogalerije, koji je prikazan na slici 5.17. Dijagram prikazuje sva sučelja fotogalerije s pripadnim funkcijama, no također prikazuje i npm pakete koji su korišteni prilikom razvoja. U ovom slučaju, paketi su odvojeni s obzirom na to pružaju li usluge za rad korisničkog sučelja ili za rad s pozadinskom podrškom.



Sl. 5.17. Dijagram komponenti razvijene web fotogalerije

Razvijena web fotogalerija se sastoji od pet Angular komponenata i 15 funkcija. Neke funkcije rade pozivajući usluge sustava BaaS, a neke se isključivo izvode na klijentskoj strani. Drugi korak analize web fotogalerije je provesti mjerenje određenih veličina koje, na pojednostavljen način, pokušavaju utvrditi složenost i veličinu web aplikacije. Prema [39], veličine koje to najprikladnije utvrđuju u domeni web aplikacija, su:

- Broj linija koda (engl. *Lines of Code, LOC*) - najosnovnija veličina, jednostavno prikazuje broj svih linija koda u nekoj datoteci.
- Broj odgovora po klasi (engl. *Response for Class, RFC*) - veličina prikazuje koliko se metoda poziva kada se poziva funkcija unutar klase. Što se više metoda poziva, veća je složenost.
- Broj lokalnih metoda (engl. *Number of Local Methods, NLM*) - veličina prikazuje broj svih lokalnih metoda u klasi ili datoteci.
- Broj udaljenih metoda (engl. *Number of Remote Methods, NRM*) - veličina prikazuje broj svih poziva udaljenih metoda u klasi ili datoteci.
- Broj atributa (engl. *Number of Attributes, NOA*) - veličina prikazuje broj atributa u klasi.
- McCabe-ov ciklometrijski model složenosti (engl. *McCabe Cyclomatic Complexity Model, CCN*) - klasična mjera složenosti izvornog koda. Prikazuje broj različitih putova kroz koje može prolaziti programski kod.
- Aferentno spajanje (engl. *Afferent Coupling, AC*) - veličina prikazuje broj klasa ili entiteta o kojima ovisi neka klasa.

U tablici 5.1, prikazani su rezultati mjerenja navedenih veličina za razvijenu web fotogaleriju. Određene veličine su promatrane na razini komponente, a ostale na razini funkcije komponente. Odabrane veličine pomažu utvrditi razinu složenosti i veličinu web aplikacije, ali i normalizirati druga mjerenja. Primjerice, vrlo visoka ciklometrijska složenost metode ukazuje na visoku složenost metode web aplikacije, no samo ukoliko metoda ima nižu vrijednost LOC veličine u odnosu na druge metode u klasi. Ovo je jedan od razloga zašto je važno prikupiti mjerenja svih navedenih veličina.

Mjerenje je provedeno uz pomoć uređivača koda Visual Studio Code. On sadrži alat Code Metrics koji na jednostavan i intuitivan način olakšava mjerenje veličina za određivanje složenosti i veličine web aplikacije. Prilikom mjerenja, za svaku komponentu su odvojeno promatrane datoteka i funkcije. Pod datotekom se podrazumijeva sav opisni kod sučelja te komponente, pisan u opisnom jeziku HTML, te programski kod logike komponente, pisan u TypeScript programskom jeziku. U mjerenje nije uključen kod oblikovanja sučelja, pisan u stilskom jeziku CSS.

Tab. 5.1. Rezultati mjerenja veličina za određivanje složenosti i veličine web fotogalerije

KOMPONENTE	FUNKCIJE	LOC	RFC	NLM	NRM	NOA	CCN	AC
AUTENTIKACIJA	DATOTEKA	244	-	2	26	13	-	10
	FUNKCIJE							
	Register	12	4	-	-	-	2	-
	Login	35	6	-	-	-	6	-

	Logout	12	5	-	-	-	2	-
	ChangePW	13	4	-	-	-	2	-
KREIRANJE ALBUMA	DATOTEKA	84	-	4	6	6	-	12
	FUNKCIJE							
	UploadAlbum	18	5	-	-	-	2	-
PRIJENOS FOTOGRAFIJE	DATOTEKA	129	-	1	12	7	-	11
	FUNKCIJE							
	UploadPhoto	25	6	-	-	-	5	-
	PreviewPhoto	21	5	-	-	-	4	-
LISTA ALBUMA	DATOTEKA	204	-	1	6	10	-	9
	FUNKCIJE							
	GetAlbums	9	2	-	-	-	2	-
	GetPhotos	11	2	-	-	-	4	-
	AlbumDetails	6	2	-	-	-	1	-
	DeleteAlbum	11	7	-	-	-	2	-
PRIKAZ FOTOGRAFIJA	DATOTEKA	211	-	2	10	11	-	9
	FUNKCIJE							
	GetAlbum	11	2	-	-	-	2	-
	GetPhotos	11	2	-	-	-	4	-
	DeletePhoto	8	5	-	-	-	2	-
	PhotoDetails	4	1	-	-	-	1	-

Treći korak analize je provođenje mjerenja istih veličina za pripadne funkcije usluga sustava BaaS, kako bi se odredio odnos složenosti i veličine ekvivalentnih funkcija korisničkog sučelja i pozadinske podrške. Funkcije koje se uzimaju u obzir su sve funkcije usluga koje se pozivaju u funkcijama sučelja razvijene web fotogalerije. Primjerice, za registriranje korisnika se promatra funkcija *Register*, a kao datoteka se promatra cijeli kontroler *AccountController*, unutar usluge upravljanja korisnicima. Ukoliko je usluga razvijena na temelju višeslojne arhitekture, u obzir se uzima i kod za pristup bazi podataka, iz nižih slojeva. Tablica 5.2 prikazuje rezultate mjerenja navedenih veličina za usluge sustava BaaS. Prilikom mjerenja, u obzir nisu uzeti programski kod modela i konfiguracijski kod usluga sustava BaaS.

Tab. 5.2. Rezultati mjerenja veličina za određivanje složenosti i veličine usluga sustava BaaS

BAAS USLUGE I FUNKCIJE	LOC	RFC	NLM	NRM	NOA	CCN	AC	FUNKCIJA SUČELJA - POZIVATELJ
USLUGA UPRAVLJANJA KORISNICIMA	656	-	4	37	8	-	22	-
FUNKCIJE								
Register	22	3	-	-	-	4	-	AUTENTIKACIJA - *Register*
Login	28	3	-	-	-	4	-	AUTENTIKACIJA - *Login*
Logout	12	2	-	-	-	2	-	AUTENTIKACIJA - *Logout*
ChangePassword	17	3	-	-	-	4	-	AUTENTIKACIJA - *ChangePW*
(User) - Get	12	3	-	-	-	2	-	LISTA ALBUMA - *GetAlbums*

								PRIKAZ FOTOGRAFIJA - *GetAlbum*
USLUGA POHRANE PODATAKA	549	-	1	10	5	-	13	-
FUNKCIJE								
(Resource) -Get	12	3	-	-	-	2	-	PRIKAZ FOTOGRAFIJA - *GetAlbum, PhotoDetails* PRIJENOS FOTOGRAFIJE - *PreviewPhoto*
(Resource) - Post	17	4	-	-	-	4	-	KREIRANJE ALBUMA - *UploadAlbum* PRIJENOS FOTOGRAFIJE - *UploadPhoto*
(Resource) - Put	17	4	-	-	-	4	-	LISTA ALBUMA - *AlbumDetails*
(Resource) - Delete	13	3	-	-	-	2	-	LISTA ALBUMA - *DeleteAlbum* PRIKAZ FOTOGRAFIJA - *DeletePhoto*
(Resource)- GetBySearchCriteria	12	3	-	-	-	2	-	LISTA ALBUMA - *GetAlbums* PRIKAZ FOTOGRAFIJA - *GetPhotos*
USLUGA UPRAVLJANJA DATOTEKAMA	384	-	5	20	6	-	17	-
FUNKCIJE								
Upload	34	9	-	-	-	6	-	PRIJENOS FOTOGRAFIJE - *UploadPhoto*
Download	14	4	-	-	-	2	-	PRIJENOS FOTOGRAFIJE - *PreviewPhoto* PRIKAZ FOTOGRAFIJA - *PhotoDetails*
Delete	14	3	-	-	-	2	-	PRIKAZ FOTOGRAFIJA - *DeletePhoto*

Nakon provedenih mjerenja veličina, koje određuju složenost i veličinu web fotogalerije i usluga sustava BaaS, moguće je izraditi usporedbu određenih mjera. Tablica 5.3 prikazuje usporedbu mjerenih veličina, na način da se svaka funkcija sučelja web fotogalerije uspoređuje s funkcijom sustava BaaS, koju ona poziva. Ukoliko neka funkcija sučelja poziva više funkcija sustava BaaS, onda se mjerene veličine tih funkcija zbrajaju. Primjerice, funkcija web fotogalerije *UploadPhoto*, za svoj rad koristi funkcije sustava BaaS: *Upload* i *(Resource) - Post*.

Tab. 5.3. Usporedba mjerenih veličina za web fotogaleriju i sustav BaaS

FUNKCIJA SUČELJA	FRONTEND / BACKEND	LOC	% LOC	RFC	% RFC	CCN	% CCN
Register	F	12	35	4	57	2	33
	B	22	65	3	43	4	67
Login	F	35	56	6	67	6	60
	B	28	44	3	33	4	40
Logout	F	12	50	5	71	2	50
	B	12	50	2	29	2	50
ChangePW	F	13	43	4	57	2	33

	B	17	57	3	43	4	67
UploadAlbum	F	18	51	5	56	2	33
	B	17	49	4	44	4	67
UploadPhoto	F	25	42	6	40	5	45
	B	34	58	9	60	6	55
GetAlbums	F	9	27	2	25	2	33
	B	24	73	6	75	4	67
GetPhotos	F	11	48	2	40	4	67
	B	12	52	3	60	2	33
AlbumDetails	F	6	26	2	33	1	25
	B	17	74	4	66	4	75
DeleteAlbum	F	11	46	7	70	2	50
	B	13	54	3	30	2	50
GetAlbum	F	11	31	2	25	2	33
	B	24	69	6	75	4	67
PhotoDetails	F	4	13	1	13	1	25
	B	26	87	7	87	4	75
DeletePhoto	F	8	23	5	42	2	33
	B	27	77	7	58	4	67
Ukupno	F	175	39	51	46	33	41
	B	273	61	60	54	48	59

U prethodnim tablicama prikazani su izmjereni rezultati za web fotogaleriju i usluge sustava BaaS. Krajnji korak analize se sastoji od istraživanja dobivenih rezultata i pokušaja izvlačenja zapažanja iz mjerenja i usporedbe složenosti i veličine ovih programskih sustava. Prva mjerena veličina od važnosti je broj linija koda u određenoj metodi. Ako se promatra zbroj svih linija koda za funkcije usluga sustava BaaS, kao i za funkcije sučelja web fotogalerije, može se zaključiti da je udio pozadinskog koda potrebnog za rad fotogalerije oko 60 % od ukupnog programskog koda. Slično vrijedi i za veličine broj odgovora po klasi (54 %) i McCabe-ov ciklometrijski model složenosti (59 %). Ako se dobiveni rezultati uzmu kao mjerodavni u određivanju veličine i složenosti promatranih sustava, ispostavilo bi se da oko 60% razvijene web fotogalerije otpada na pozadinsku podršku. S obzirom da za razvoj web fotogalerije nije potrebno pisati nimalo programskog koda za pozadinsku podršku, njena veličina i složenost su prema tome umanjene za oko 60%. Također, isto vrijedi i za vrijeme razvoja klijentske web aplikacije, ako se pretpostavi da se pozadinska podrška i korisničko sučelje razvijaju istom brzinom.

6. ZAKLJUČAK

Tema ovog rada bila je prikazati razvoj vlastitog sustava BaaS i analizirati prednosti upotrebe takvog sustava prilikom razvoja web aplikacija. Analizirajući nekoliko poznatijih sustava tog tipa, zaključeno je da se sustav BaaS sastoji od skupa raznih usluga, koje se u velikoj mjeri ponavljaju, a karakteristične su za domenu usluga web i mobilnih aplikacija. Tako su u ovom radu odabrane tri usluge koje se pojavljuju u svim analiziranim sustavima toga tipa, a neophodne su za rad većine današnjih web i mobilnih aplikacija. Odabrane usluge su: pohrana podataka, upravljanje korisnicima te upravljanje datotekama. S obzirom na koncept sustava BaaS, kao skup usluga, sustav je razvijen na temelju arhitekture mikrousluga. U kontekstu odabira arhitekture ovakvog sustava, važna napomena je da arhitektura mikrousluga nije usko povezana s pojmom sustava BaaS te odabir arhitekture uvelike ovisi o preferencijama arhitekta i razvojnih programera, kao i o poslovnim zahtjevima. S obzirom na provedenu analizu već postojećih sustava BaaS, može se iznijeti zaključak kako je arhitektura takvog sustava u stvarnosti mnogo složenija od osnovne arhitekture mikrousluga. Složenost nastaje zbog potreba upravljanja, verzioniranja, filtriranja prometa, rukovanja s iznimkama, zapisivanja pogrešaka, provođenja sigurnosnih pravila, korištenja priručne memorije i drugih.

Kao posrednik između sustava BaaS i web aplikacije, razvijen je SDK koji olakšava povezivanje web aplikacije s pozadinskim uslugama sustava BaaS te sakriva njegovu internu strukturu od krajnjeg korisnika. SDK je usko vezan uz programski jezik ili razvojni okvir, pa bi stoga trebalo razviti širok spektar SDK-ova, kako bi razvijeni sustav BaaS postao šire korišten. U konačnoj analizi veličine i složenosti klijentske web aplikacije, prikazano je da upotreba sustava BaaS veoma olakšava i ubrzava njen razvoj. Ako se dobiveni rezultati uzmu kao mjerodavni u određivanju veličine i složenosti promatranih komponenti, ispostavilo bi se da oko 60% razvijene web aplikacije otpada na pozadinsku podršku. Prilikom korištenja rezultata ove analize, bitno je imati na umu da ona ne vrijedi općenito za sve web aplikacije koje potencijalno mogu koristiti neki sustav BaaS. Rezultati analize su vrlo usko vezani uz različite parametre razvoja web aplikacije, kao što su: veličina, tip i funkcionalnosti aplikacije, razvojni okvir, programski jezik, znanje, iskustvo i preferencije programerskog tima, arhitektura aplikacije i drugi. Također, rezultati analize ovise i o sustavu BaaS koji se koristi kao pozadinska podrška. Uvjeti u kojima se rezultati provedene analize mogu smatrati mjerodavnima, podrazumijevaju klijentsku aplikaciju sličnih funkcionalnosti i programskog koda kao razvijena web fotogalerija i sustav BaaS slične arhitekture i programskog koda kao razvijeni sustav BaaS u ovom radu. Provedena analiza može

biti referentna za ispitivanje prednosti sustava BaaS, no korisnik analize mora znati ocijeniti važnost i težinu utjecaja spomenutih parametara razvoja na rezultate analize.

U konačnici se mogu izdvojiti glavni izazovi koji se nameću prilikom razvoja sustava BaaS. Ti izazovi su: različitost klijenata, automatska skalabilnost, istovremenost, konkurentnost, visoke performanse i sigurnost. Razvijeni sustav BaaS ima još prostora za daljnji napredak, uzimajući u obzir analizu postojećih takvih sustava. Očigledno, prvi prijedlog za širenje sustava je razvoj još nekoliko usluga karakterističnih za domenu web aplikacija. Ipak i postojeće usluge se mogu dodatno poboljšati. Na prvom mjestu prijedloga za poboljšanje je uvođenje korisničkih uloga te popisa za kontrolu pristupa (engl. *access control list*) koji specificira koji korisnici dobivaju pristup kojim objektima. Nadalje, idući prijedlog za poboljšanje je uvođenje dodatnih sigurnosnih mehanizama, kao što su detekcija neželjenih zahtjeva, konfiguriranje protokola HTTPS i enkripcija podataka. Na kraju je moguće zaključiti da je BaaS samo jedan od mnogih modela računarstva u oblaku koji mijenjaju način razvoja aplikacija, a prvenstveno u svrhu bržeg, jednostavnijeg i pouzdanijeg razvoja. Prilikom odlučivanja za upotrebom sustava BaaS u razvoju web ili mobilne aplikacije, donositelj odluke mora imati temeljno znanje o tim sustavima, koje je prezentirano u ovom radu.

LITERATURA

[1] M. Monroe, „The Gospel of MBaaS (Part 1 of 2)“, InfoQ, 2013., https://www.infoq.com/news/2013/05/MBaaS-Anypresence, pristupljeno: 15.5.2018.

[2] „What is Cloud Computing?“, Amazon, 2013., https://aws.amazon.com/what-is-cloud-computing/, pristupljeno: 15.5.2018.

[3] P. Mell, T. Grance, „Computer Security“, The NIST Definition of Cloud Computing, No. 800, Vol. 145., pp. 1-7, September, 2011.

[4] J. Clark, „What is backend as a service?“, The Register, San Francisco, 2013., https://www.theregister.co.uk/2013/08/17/firebase_baas_ga/, pristupljeno: 20.5.2018.

[5] A. Ashwini, „How to choose best MBaaS“, A Medium Corporation, US, 2017., https://medium.com/swlh/how-to-choose-the-best-mobile-backend-as-a-service-mbaas-5534e1fc33f4, pristupljeno: 21.5.2018.

[6] R. Lawler, „Parse offers its mobile backend-as-a-service on a freemium model“, GigaOM, 2012., https://gigaom.com/2012/03/26/parse-freemium-service/, pristupljeno: 22.5.2018.

[7] D. Nations, „Web Applications“, About.com, 2018., https://www.lifewire.com/what-is-a-web-application-3486637, pristupljeno: 24.5.2018.

[8] B. Carter, „Grow your own Backend-as-a-Service platform“, Conference College of Information & Computer Technology, Vol. 2015., pp. 1-6, 2016.

[9] G. Batschinski, „What are pros and cons of building WEB app with BaaS?“, Quora, 2016., https://www.quora.com/What-are-the-pros-and-cons-of-building-WEB-app-with-BaaS, pristupljeno: 26.5.2018.

[10] T. Huston, „What is Microservice Architecture?“, Smartbear, 2018, https://smartbear.com/learn/api-design/what-are-microservices/, pristupljeno: 1.6.2018.

[11] „What is REST?“, Codeacademy, 2018., https://www.codecademy.com/articles/what-is-rest, pristupljeno: 1.6.2018.

[12] R.T. Fielding, „Architectural Styles and the Design of Network-based Software Architectures“, University of California, Irvine, 2000.

[13] „A Modern Reintroduction To AJAX“, JavaScriptCoder, 2017., http://javascript-coder.com/tutorials/re-introduction-to-ajax.phtml, pristupljeno: 2.6.2018.

[14] „The JSON Data Interchange Standard“, Ecma-International, 1999., https://www.json.org/, pristupljeno: 2.6.2018.

[15] J. Reinke, „Understanding OAuth2“, BubbleCode, 2016., http://www.bubblecode.net/en/2016/01/22/understanding-oauth2/, pristupljeno: 13.6.2018.

[16] L. Trieloff, „Three Principles of API First Design“, Medium, 2017., https://medium.com/adobe-io/three-principles-of-api-first-design-fa6666d9f694, pristupljeno: 2.6.2018.

[17] S. Clarke, „Measuring API Usability“, Dr.Dobb's, 2004, http://www.drdobbs.com/windows/measuring-api-usability/184405654, pristupljeno: 2.6.2018.

[18] „Deployment Patterns“, Microsoft, 2014., https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff646997(v=pandp.10), pristupljeno: 3.6.2018.

[19] K. Horvat, „Multi-layered Architectures in .NET“, Mono, 2017., http://www.mono.hr/pdf/Multi-Layered%20Architectures%20in%20.Net-Web.pdf, pristupljeno: 3.6.2018.

[20] J.,Shore, „Dependency Injection Demystified“, I.T.Titanium, California, 2006., http://www.jamesshore.com/Blog/Dependency-Injection-Demystified.html, pristupljeno: 4.6.2018.

[21] R.E. Johnson, B. Foote, „Designing Reusable Classes“, Journal of Object-Oriented Programming, Vol. 1, No. 2, pp. 1-8, August, 1991.

[22] „Innovate Like a Startup, Deliver for an Enterprise“, Progress, 2014., https://www.progress.com/kinvey/platform, pristupljeno: 25.5.2018.

[23] „Enterprise BAAS“, Backendless, 2013., https://backendless.com/what-is-backend-as-a-service/enterprise-baas/, pristupljeno: 25.5.2018.

[24] „Parse Case Study“, Amazon, 2012., https://aws.amazon.com/solutions/case-studies/parse/, pristupljeno: 25.5.2018.

[25] „Introduction to the C# Language and .NET Framework“, Microsoft, 2015., https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework, pristupljeno: 4.6.2018.

[26] „Visual Studio 2017 version 15.7 Release Notes“, Microsoft, 2018., https://docs.microsoft.com/en-us/visualstudio/releasenotes/vs2017-relnotes#15.7.0, pristupljeno: 4.6.2018.

[27] „ASP.NET Overview“, Microsoft, 2010., https://docs.microsoft.com/en-us/aspnet/overview#web-apis, pristupljeno: 4.6.2018.

[28] J. Chadwick, T. Snyder, H. Panda, „Razvoj Web aplikacija uz ASP.NET MVC“, IT Expert, Zagreb, 2012.

[29] „What is MongoDB“, MongoDB, 2009., https://www.mongodb.com/what-is-mongodb, pristupljeno: 5.6.2018.

[30] „Ninject Documentation“, GitHub - Ninject, 2018., https://github.com/ninject/Ninject, pristupljeno: 5.6.2018.

[31] J. Galloway, P. Rastogi, R. Anderson, T. Dykstra, „Introduction to ASP.NET Identity“, Microsoft, 2013., https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity, pristupljeno: 13.6.2018.

[32] „What is Entity Framework“, Entity Framework Tutorial, 2013., http://www.entityframeworktutorial.net/what-is-entityframework.aspx, pristupljeno: 13.6.2018.

[33] M. Waason, „Secure a Web API with Individual Accounts and Local Login in ASP.NET Web API 2.2“, Microsoft, 2014., https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/individual-accounts-in-web-api, pristupljeno: 15.6.2018.

[34] „SQL Server Documentation“, Microsoft, 2018., https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation?view=sql-server-2017, pristupljeno: 19.6.2018.

[35] „Introduction to Object Storage in Azure“, Microsoft, 2018., https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction, pristupljeno: 25.6.2018.

[36] C. Richardson, „Pattern: API Gateway / Backend for Front-End“, Microservices.io, 2016., http://microservices.io/patterns/apigateway.html, pristupljeno: 27.6.2018.

[37] „What is Angular?“, Angular, 2010., https://angular.io/docs, pristupljeno: 3.7.2018.

[38] „What is npm?“, npmjs.com, 2018., https://docs.npmjs.com/getting-started/what-is-npm, pristupljeno: 13.7.2018.

[39] C.A. McNinch, „Measuring and Quantifying Web Application Design“, Graduate Student Theses, Dissertations, Professional Papers, Vol. 657., 2012.

SAŽETAK

Cilj ovog diplomskog rada je analiza, dizajn i implementacija vlastitog sustava BaaS za web aplikacije. Sustav BaaS je jedan od uslužnih modela računarstva u oblaku, čija je svrha olakšati i ubrzati razvoj web i mobilnih aplikacija. Glavni nedostatak uslužnog modela BaaS je velika ovisnost o pružatelju sustava, stoga ovaj diplomski rad razmatra razvoj vlastitog sustava BaaS. Analiza sustava BaaS izvedena je usporedbom arhitektura postojećih sustava, na temelju koje su izvedeni osnovni arhitekturni koncepti i stilovi koji se koriste prilikom razvoja sustava BaaS u ovom radu. U fazi dizajna, odlučeno je da se sustav BaaS temelji na arhitekturi mikrousluga, a sadrži tri usluge karakteristične za današnje web aplikacije: pohrana podataka, upravljanje korisnicima te upravljanje datotekama. Implementacija svih usluga ostvarena je uz pomoć razvojnog okvira ASP.NET Web API, a svaka usluga sadrži vlastiti sustav za pohranu podataka. U svrhu jednostavnije upotrebe sustava BaaS, razvijen je SDK u razvojnom okviru Angular. Na kraju rada prikazana je izrada web aplikacije kroz razvijeni SDK i provedena analiza složenosti i veličine iste. Ishod provedene analize upućuje, da se kao posljedica upotrebe sustava BaaS u razvoju web aplikacije, umanjuju njena veličina i složenost za oko 60%. U konačnici je donesen zaključak o izazovima razvoja sustava BaaS i prednostima koje se time ostvaruju.

Ključne riječi: arhitektura mikrousluga, BaaS, računarstvo u oblaku, RESTful API, SDK.

ABSTRACT

The aim of this thesis is to analyze, design and implement BaaS system for web applications. The BaaS system is a cloud computing service model, whose purpose is to facilitate and accelerate development of web and mobile applications. The main drawback of the BaaS service model is less control due to vendor lock-in, therefore this thesis is considering the development of its own BaaS system. The analysis was performed by comparing the architecture of the existing systems, based on which the basic architectural concepts and styles used to develop the BaaS system in this paper were derived. In the design phase, it was decided that the BaaS system is based on microservice architecture, and it offers three common services of today's web applications: data storage, user management and file management. The implementation of all services has been accomplished using the ASP.NET Web API framework. Also, each service contains its own data storage system. For simpler usage of BaaS system, SDK has been developed in the Angular Framework. At the end of the paper, a web application was developed through the SDK and a complexity and size analysis was performed. The analysis shows that because of using the BaaS system in the development of the web application, its size and complexity are reduced by about 60%. Lastly, a conclusion was reached on the challenges of the BaaS system development and the benefits that have been achieved.

Keywords: microservice architecture, BaaS, cloud computing, RESTful API, SDK.

ŽIVOTOPIS

Mario Dudjak je rođen u Našicama 25. ožujka 1995. godine gdje je završio osnovnu školu i prirodoslovno – matematičku gimnaziju. Preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku upisao je 2013. godine. 2016. godine stječe akademski naziv prvostupnik inženjer računarstva te upisuje diplomski studij Programskog inženjerstva na istom fakultetu. Tijekom studija sudjeluje na raznim natjecanjima u području računarstva, poput IEEE Extreme, Elektrijada i IEEE MADC. U dva navrata je nagrađen dekanovim priznanjem za uspjeh tijekom studija. Uz fakultet, polazi i završava PHP akademiju organiziranu od strane fakulteta i tvrtke Inchoo. Također, uspješno stječe sve certifikate u okviru programa *Microsoft Professional Degree Data Science*. Na drugoj godini diplomskog studija odrađuje stručnu praksu u tvrtki Mono. Nakon završetka stručne prakse, zapošljava se u istoj tvrtki na poziciji *Software developer*. Suautor je znanstvenog rada pod nazivom *Survey of Database Backup Management,* koji je objavljen u zborniku Međunarodnog znanstveno stručnog skupa OTO. Aktivno se služi engleskim jezikom u govoru i u pismu te ima osnovno poznavanje slovačkog i njemačkog jezika.

POPIS UPOTRJEBLJENIH KRATICA

BaaS - Backend as a Service

MBaaS - Mobile Backend as a Service

API - Application Programming Interface

SDK - Software Development Kit

IaaS - Infrastructure as a Service

PaaS - Platform as a Service

SaaS - Software as a Service

REST - REpresentational State Transfer

URI - Uniform Resource Identifier

JSON - JavaScript Object Notation

XML - EXtensible Markup Language

SOAP - Simple Object Access Protocol

HTML - HyperText Markup Language

CSS - Cascading Style Sheets

HTTP - HyperText Transfer Protocol

HTTPS- HyperText Transfer Protocol Secure

MIME - Multipurpose Internet Mail Extensions

DI - Dependency Injection

IoC - Inversion of Control

SQL - Structured Query Language

NoSQL - Non SQL

MSSQL - Microsoft SQL Server

CLR - Common Language Runtime

CLI - Common Language Infrastructure

IL - Intermediate Language

MVC - Model View Controller

BSON - Binary JSON

CRUD - Create, Read, Update and Delete

OWIN - Open Web Interface for .NET

POCO - Plain Old CLR Object

ACID - Atomicity, Consistency, Isolation and Durability

TB - Terabyte

PRILOZI (na CD-u)

Prilog 1. Diplomski rad u docx formatu

Prilog 2. Diplomski rad u pdf formatu

Prilog 3. Programski kod