

**INTERNACIONALNI UNIVERZITET TRAVNIK U
TRAVNIKU**
**FAKULTET POLITEHNIČKIH NAUKA TRAVNIK U
TRAVNIKU**

ZAVRŠNI RAD

**Backend as a Service sistemi web aplikacija kroz
programski jezik Python**

Mentor:
Prof.dr. Mladen Radivojević

Student:
Nedim Hozić

SADRŽAJ

1.	UVOD	4
2.	BACKEND AS A SERVICE	5
2.1.	Pojam backenda	5
2.2.	Pojam BaaS-a	6
2.3.	Svrha BaaS sistema	6
2.4.	Uslužitelji BaaS sistema	7
2.4.1.	Appcelerator	7
2.4.2.	Appery.io.....	8
2.4.3.	Kinvey.....	8
2.4.4.	Parse	8
3.	ARHITEKTURA BAAS SISTEMA	8
3.1.	Uobičajeni blokovi BaaS sistema	8
3.1.1.	Korisnici	8
3.1.2.	Upravljanje sadržajem	9
3.1.3.	Podaci	9
3.1.4.	Pretraživač podataka	9
3.1.5.	Ključ – vrijednost	9
3.1.6.	MySql	9
3.1.7.	Relacije	9
3.1.8.	Tabele	9
3.1.9.	Slike i fotografije	9
3.1.10.	Novi integrisani kod	10
3.1.11.	Novi integrisani objekti.....	10
3.1.12.	Novi integrisani API-ji	10
3.1.13.	API upiti	10
3.1.14.	REST API.....	10
3.1.15.	Virtuelna trgovina	10
3.1.16.	Monetizacija i promocije	10
3.1.17.	Ocenjivanje	10
3.1.18.	Chat komunikacija.....	11
3.1.19.	Email.....	11
3.1.20.	Drugi sistemi za razmjenu poruka.....	11
3.1.21.	Obavijesti (engl. <i>push notifikacije</i>)	11
3.1.22.	SMS.....	11

3.1.23. Lokacija	11
3.1.24. Mjesta	11
3.1.25. Ciljanje	11
3.1.26. Igre	12
3.1.27. Dostupnost i skaliranje	12
3.1.28. Android	12
3.1.29. Blackberry	12
3.1.30. iOS	12
3.1.31. HTML5	12
3.1.32. Podrška za kompanije	12
4. PREDLOŽENO PROGRAMSKO RJEŠENJE – BAAS SISTEM KROZ PROGRAMSKI JEZIK PYTHON	12
4.1. Opis korištenih tehnologija	13
4.1.1. Programski jezik Python	13
4.1.2. PyCharm	14
4.1.3. Flask	14
4.1.4. SQLite	14
4.1.5. JavaScript programski jezik	15
4.1.6. JSON format podataka	15
4.1.7. Heroku platforma	15
4.2. Implementacija	16
4.2.1. Implementacija REST API-ja u programskom jeziku Python	16
4.2.2. Implementacija SDK-a	25
4.2.3. Implementacija klijentske web aplikacije	27
ZAKLJUČAK	33
LITERATURA	34

1. UVOD

U pozadini svakog programskog rješenja, odnosno sistema programskih rješenja postoji niz pozadinskih usluga (engl. *backend*) koje korisnici sistema ne vide, ali uveliko koriste preko programskog korisničkog interfejsa, ili prednjeg sučelja (engl. *frontend*). Posao stvaranja takvih sistema je obilan i vremenski iscrpljujući zadatak, te se naručiocu nerijetko odlučivaju za gotova postojeća rješenja u cilju uštade resursa i vremena. Takva gotova rješenja u većini slučajeva sadrže sličan skup pozadinskih funkcija, nevezano za oblast djelovanja samog sistema, te ih mogu koristiti programeri web i mobilnih aplikacija. Trenutno najmoderniji pristup razvoju takvih sistema su BaaS sistemi (engl. *Backend as a Service*) koji omogućuju programerima klijentskih web i mobilnih aplikacija da koriste isti set usluga u pozadini. Upravo zbog brzine rada sistema, te relativno kratkog vremena za razvoj novih funkcionalnosti iznimno je popularan trend među IT zajednicom.

U prvom dijelu ovog rada definisani su osnovni pojmovi potrebni za razumijevanja BaaS-a, te detaljnije objašnjenje šta je to ustvari BaaS, kada se koristi, te koje su prednosti korištenja ovih sistema. Osim toga, definisani su i neki od najpopularnijih uslužitelja BaaS-a. U drugom dijelu ulazi se malo dublje u samu materiju, te se definišu pojmovi koji su vezani za samu arhitekturu ovih sistema, koji su blokovi gradivno tkivo BaaS-a, te koja je njihova namjena.

U posljednjem, trećem dijelu rada akcenat je na izradi jednog jednostavnog BaaS sistema koji sadrži samo osnovne stvari, odnosno dijelove koji čine jedan BaaS sistem. Definisan je backend sistema kao REST API, uz odgovarajući paket za razvoj programa (engl. *Software Development Kit - SDK*), te web aplikaciju koja komunicira sa backendom preko datog SDK-a.

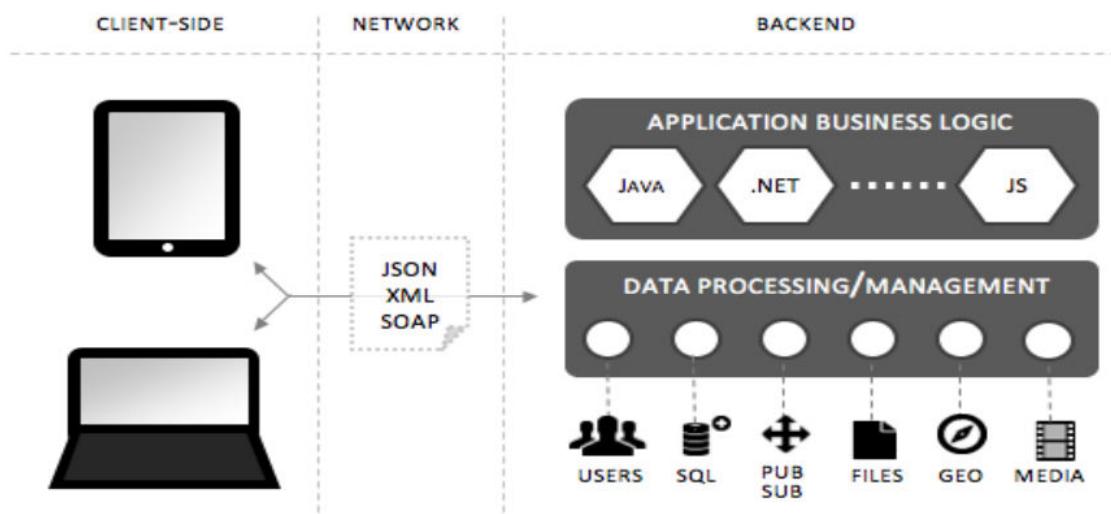
2. BACKEND AS A SERVICE

U ovom poglavlju prezentovane su osnove BaaS sistema, takođe poznatog i kao Mobile backend as a Service (skr. *MBaaS*). Ono što je specifično za ove sisteme je da se mogu posmatrati kao relativno novi model u današnjem programiranju u oblaku (engl. *cloud*), i to od 2011. godine. Osim BaaS-a, sistemi bazirani na istom modelu pružanja usluga na zahtjev su i IaaS (engl. Infrastructure as a Service), PaaS (engl. Platform as a Service), te SaaS (engl. Software as a Service).

U ovom poglavlju definisane su osnove i pojma BaaS-a, svrha, te razlike između pojedinih sistema koji su bazirani na istim ili sličnim modelima.

2.1. Pojam backenda

Da bi razumijeli šta je BaaS, prvo ćemo definisati pojma backend-a. Naredna slika prikazuje šta je ustvari backend.



Slika 1. Backend¹

Kao što vidimo sa Slike 1. backend se sastoji od aplikacijske biznis logike i sektora za obradu i procesiranje podataka koji je dalje podijeljen na sektore za korisnike, bazu podataka (SQL), pub/sub (engl. *publish-subscribe pattern* – za slanje i primanje poruka), datoteke, lokacije, te medija (muzika, filmovi, slike...).

Sa klijentskom stranom backend može da komunicira preko JSON i XML datoteka (HTTP protokol), te SOAP protokolom.

2.2. Pojam BaaS-a

Nakon što smo definisali sve potrebne pojmove, u ovom poglavlju ćemo objasniti BaaS.

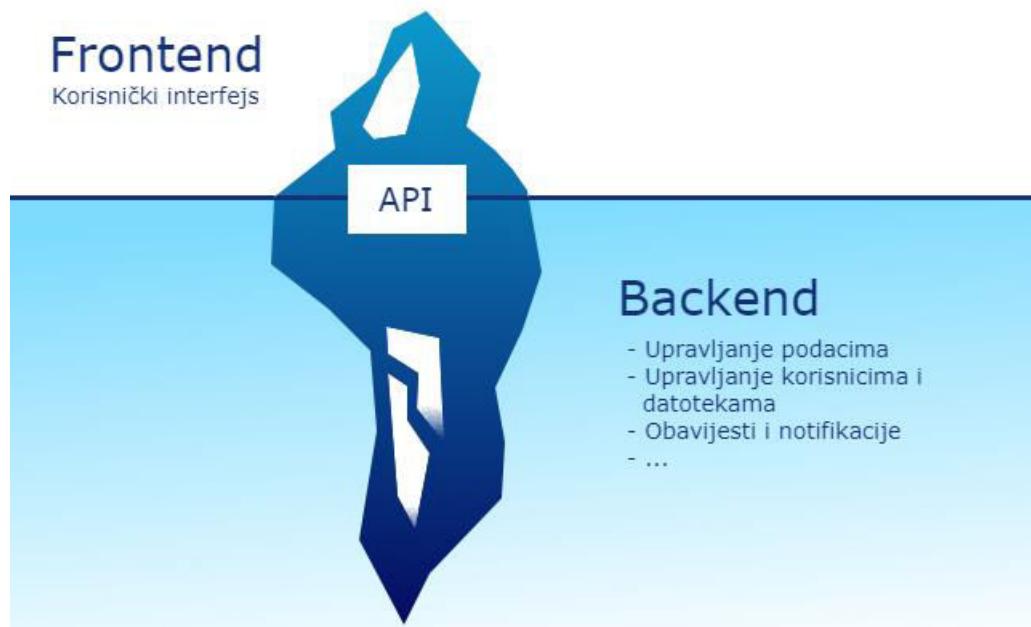
¹ Nguyen, P.: *Mobile Backend as a Service: The Pros and Cons of Parse*, Lahti University of Applied Sciences, Lahti, 2016

BaaS je usluga u oblaku koja služi kao posrednički softver koji omogućava programerima načine za povezivanje web i mobilnih aplikacija na cloud usluge putem aplikacijskog programskog interfejsa (engl. *API – Application Programming Interface*, skup rutina, funkcija, protokola i alata koje služe pri razvoju aplikacija), te SDK-a (set jednog ili više API-ja, programskih alata, te dokumentacije koja dozvoljava programeru da ustvari programira u određenoj platformi).²

2.3. Svrha BaaS sistema

Funkcionalnosti BaaS-a uključuju pohranu podataka u oblaku, obavijesti i notifikacije, upravljanje poslužiteljem (engl. *server*), upravljanje korisnicima i datotekama, integracija u društvene mreže, usluge lokacije, te mnoge druge usluge. Usluge BaaS sistema imaju vlastiti API, te olakšava programerima jednostavnu integraciju u aplikacije.³

Web i mobilne aplikacije imaju slične ili iste funkcionalnosti iza ekrana. Svaka od tih funkcionalnosti ima svoj API koji mora biti povezan sa svakom frontend aplikacijom pojedinačno. BaaS sistem, u ovom slučaju, pruža jedinstvenu vezu klijentskih aplikacija sa frontend dijelom BaaS-a, te dalje na backend dio.



Slika 2. Prikaz odnosa backenda i frontenda

Dakle, glavne funkcionalnosti cjelokupnog sistema su ustvari isprogramirane jednom, na strani BaaS sistema, te iste funkcionalnosti koriste sve klijentske aplikacije, bilo Android, iOS ili web aplikacije. Kod tradicionalnog načina programiranja, odnosno koncepata, to nije bio slučaj, pa je svaka klijentska aplikacija imala i svoj backend, te se ustvari radio „duplic posao“. Ovo su samo neke od benefita BaaS sistema:

² Rasthofer, S., Arzt, S., Hahn, R., Kolhagen, M., Bodden, E.: (*In*)Security of Backend-as-a-Service, Center for Advanced Security Research Darmstadt, Darmstadt, 2016

³ <https://www.techopedia.com/definition/29428/backend-as-a-service-baas> (15.06.2019.)

❖ Dostupnost

Kao što smo već naveli, sa BaaS-om lako povezujemo aplikacije sa različitim platformi, a to doprinosi i mnogim drugima prednostima, kao što je jednostavnija razmjena podataka, pristup informacijama sa bilo kojeg mesta, konzistentnost, te bolje korisničko iskustvo.

❖ Mogućnost stvaranja različitosti iz jednog modela

Svaki korisnik sistema, u početnom trenutku, u mogućnosti je da koristi istu bazu sistema. Pošto su sve jedinke tog sistema, za sve korisnike u početnom trenutku iste, svaki korisnik ima mogućnost da „proširi“ polje djelovanja samog sistema, pa i prekvalificuje sam sistem.

❖ Nema ponovnog implementiranja istih stvari

Za razliku od tradicionalnih pristupa, programeri nisu prisiljeni da ponovo stvaraju, odnosno programiraju identične stvari i funkcionalnosti za svaku od platformi aplikacije. Umjesto što bi trošili vrijeme na to, dužni su samo da povežu klijentsku stranu aplikacija na API BaaS sistema preko SDK-a, te nema bespotrebnog razvijanja identičnih stvari i funkcionalnosti. Programer se može usredosrediti na izgradnju onoga što mu je potrebno, te raditi određene nadogradnje, umjesto da počinju od nule svaki put, za svaku platformu aplikacije.⁴

2.4. Uslužitelji BaaS sistema

Svaki BaaS sistem je različit u određenom smislu, pa tako i svaki od njih zahtjeva različite postavke samog sistema, platforme, operativne sisteme i sl. Osim toga, prilikom same izrade projekta vodi se računa i o uslužiteljima budućeg BaaS sistema. Nema striktne definicije BaaS uslužitelja, jer sam taj pojam varira od okolnosti, zahtjeva, ljudstva i sl.

U današnje vrijeme postoji veliki broj BaaS uslužitelja, međutim prilikom istraživanja raznih članaka, blogova i natpisa, možemo sa sigurnošću navesti neke lidera u tom polju.⁵

2.4.1. Appcelerator

Appcelerator je jedan od lidera u polju BaaS-a za enterprise sisteme, te je kompaktibilan sa više platformi. Trenutno usluge Appcelerator-a koristi više od 1,6 miliona programera sa preko 30 000 povezivanja na web, mobilne i desktop aplikacije na 30 miliona uređaja svaki dan. Prema zvaničnoj web stranici, korisnici Appcelerator-a mogu iskoristiti svoje postojeće vještine, te na jednostavan način postaviti temelje velikih sistema, te uživati konstantnu fleksibilnost i kontrolu.⁶

2.4.2. Appery.io

Appery.io je prva mobilna platforma koja nudi razvojno okruženje na oblaku sa prethodno definisanim backend uslugama, te izuzetno moćnim alatom za organizaciju i praćenje rada

⁴ <https://telusdigital-marketplace-production.s3.amazonaws.com/iot/user-content/product/818d-o.pdf>
(Preuzeto 14.06.2019.)

⁵ Lane, K.: *Overview of the Backend as a Service (BaaS) Space*, API Evangelist, 2013

⁶ <https://www.appcelerator.com/> (17.06.2019.)

timova. Ono što je bitno spomenuti za ovu platformu je i prilično dobra tehnička podrška, te se razvojni timovi mogu fokusirati na druge stvari ukoliko se problem pojavi na strani platforme.⁷

2.4.3. Kinvey

Kinvey je jedna od najjednostavnijih platformi u smislu postavljanja početnih opcija i postavki same aplikacije. Krasiti je prilično velika fleksibilnost, a sudeći prema raznim komentarima na društvenim mrežama, programeri su prilično zadovoljni stabilnošću, te tehničkim karakteristikama Kinvey platforme.⁸

2.4.4. Parse

Jedna od najčešće spominjanih platformi u krugu IT inžinjera je sigurno Parse platforma. Ako je suditi prema zvaničnoj dokumentaciji ova platforma sadrži veliki broj SDK-a za sve vrste aplikacija, počevši od iOS, Android, Windows, OS X i JavaScript.⁹

3. ARHITEKTURA BAAS SISTEMA

Svaki BaaS sistem sastoji se od blokova, odnosno modula koji su pojedinačno zaduženi za određeni set funkcionalnosti, te mogu biti zavisni i nezavisni. Zavisni blokovi nisu u mogućnosti da obavljaju funkcije sami, nego u interakciji sa drugim blokovima daju željeni rezultat. Za razliku od njih, nezavisni blokovi mogu postojati sami za sebe, te im nisu potrebni drugi blokovi i moduli da bi funkcionalisali. Dobra praksa prilikom izrade BaaS sistema, a posmatrajući tehničku stranu, je pokušati definisati što više nezavisnih blokova, ali bez ponavljanja određenih implementacija, te učiniti sistem pogodnim za prekvalifikaciju, odnosno uz minimalne prepravke učiniti da bude pogodan i za neke druge oblasti. Jedan od primjera može biti sistem za agente nekretnine, koji ako se definiše „uredno“, može kasnije poslužiti, uz minimalne izmjene, za neke druge oblasti rada, recimo salona za prodaju automobila.

3.1. Uobičajeni blokovi BaaS sistema

Svaki BaaS sistem ili platforma ima neke zajedničke blokove koji se koriste, a čak neki uslužitelji, po osnovu, pružaju već definisane ove blokove.¹⁰

3.1.1. Korisnici

Najosnovniji dio svakog sistema – korisnici. Osnovni modul za upravljanje korisnicima omogućava da se korisnici registruju, prijave, te koriste određeni sistem. Za ovaj modul kaže se da je jezgro svakog BaaS sistema.

3.1.2. Upravljanje sadržajem

Modul, odnosno blok za upravljanjem sadržaja na aplikaciji omogućava kreiranje, ažuriranje i brisanje stranica, sadržaja, te generalno strukture sajta. Pristupi u ovom modulu

⁷ <https://appery.io/> (17.06.2019.)

⁸ <https://www.progress.com/kinvey> (18.06.2019.)

⁹ <http://parseplatform.org/> (18.06.2019.)

¹⁰ Lane, K.: *Overview of the Backend as a Service (BaaS) Space*, API Evangelist, 2013

mogu biti različiti, ali tendencioznost je da bude što sličniji Web CMS-u (engl. *Content Management System*), ali je ustvari jednostavniji, te pogodan za mobilne uređaje između ostalog.

3.1.3. Podaci

Bez podataka ni jedan sistem nema smisla. Podaci se nalaze u svakom sistemu, bili oni ulazni ili izlazni. U nastavku su navedeni samo neki od blokova koji su vezani za podatke, a posjeduje ih većina uslužitelja.

3.1.4. Pretraživač podataka

Pretraživači podataka (engl. *data browsers*) pružaju brz, ali neorganizovan način pregleda podataka koje se koriste u backendu sistema. Mogu da budu potpuno jednostavni, ali i kompleksni sa pripadajućim interfejsima za pregled podataka na većim sistemima.

3.1.5. Ključ – vrijednost

Ključ – vrijednost (engl. *key – value*) omogućava aplikacijama da spremaju podatke bez ikakve sheme, odnosno podaci se spremaju u tipu podataka nekog programskog jezika ili objekta.

3.1.6. MySql

MySQL je jedna od popularnijih baza podataka, a kao izvor podataka koriste je mnoge web, ali i mobilne aplikacije. MySQL je uobičajen konektor prilikom razvoja BaaS sistema za pristup i obradu podataka.

3.1.7. Relacije

Relacijske tabele omogućavaju povezivanje tabličnih podataka u zasebne tablice, a istovremeno omogućuje da polja i njihove vrijednosti budu povezane relacijama. Pružanje mogućnosti programerima za razvoj relacija među podacima je uobičajena karakteristika BaaS sistema. Obično su relacijske tabele razvijene na platformama kao što je MySQL.

3.1.8. Tabele

Omogućava programerima da čuvaju podatke u kolonama i redovima. Pružanje jednostavnog načina za spremanje tekstualnih, numeričkih, te mnogih drugih tipova podataka je od suštinskog značaja za bilo koji backend, ali i osnovni zahtjev za sve BaaS uslužitelje.

3.1.9. Slike i fotografije

Mogućnost da se upravlja slikama i fotografijama je esencijalna funkcionalnost svake današnje web i mobilne aplikacije. Blokovi za pohranu slika su uobičajena funkcionalnost svakog BaaS sistema.

3.1.10. Novi integrисани kod

Svaki BaaS sistem radi u određenom okruženju, te dozvoljava programerima da rapidno razvijaju i programiraju aplikacije, ali isto tako potrebno je proširiti ili prilagoditi

funkcionalnosti određenim kodom. Mnoge BaaS platforme dozvoljavaju injektovanje određenog koda za prilagodbu unutar kontroliranih dijelova platforme, te se taj kod izvršava zajednom sa onim ugrađenim.

3.1.11. Novi integrisani objekti

Prilagođeni objekti omogućuju programerima da kreiraju vlastitu strukturu podataka (tj. klase i objekte) koji im dozvoljavaju da bilo koji podatak ili sadržaj čuvaju u ovim objektima. Programeri mogu da kreiraju bilo koji objekat sa nekoliko ključnih karakteristika koje dozvoljavaju kontrolu i manipulaciju ovim objektima unutar svojih programa i aplikacija.

3.1.12. Novi integrisani API-ji

Nekoliko BaaS platformi obezbeđuje alate za programere da uvedu prilagođene API-je koji rade sa prilagođenim blokovima za izradu objekata, omogućavajući im da prošire API izvan unaprijed definisanih okvira. Proširivanje platformi pomoću prilagođenih API-ja je brz način za proširenje izvan početnih BaaS funkcionalnosti.

3.1.13. API upiti

Mnoge BaaS platforme nude interfejs koji služi za kreiranje upita, a koje dozvoljavaju SQL pristup, pa pomaže programerima da direktno rade sa podacima. Programski interfejsi za upite su uveliko poznati programerima, kao i interfejsi za REST upite.

3.1.14. REST API

Sve BaaS platforme pružaju API za programere da bi mogli jednostavno dobiti uvid u svaki aspekt aplikacije. API-ji omogućavaju upotrebu u izgradnji aplikacija, kao i integraciju sa drugim sistemima upravljanja.

3.1.15. Virtuelna trgovina

Trgovina unutar aplikacije (engl. *In-App Purchase*) je neophodna za današnje programere web i mobilnih aplikacija. Uobičajeni pristup većine BaaS platformi je omogućavanje programerima da implementiraju kupovinu u aplikacijama koristeći virtuelne valute.

3.1.16. Monetizacija i promocije

Kako bi se podržale druge strategije monetizacije, poticaja i lojalnosti unutar aplikacija, tvorci BaaS-a grade alate za promociju, koji razvojnim inženjerima pružaju mogućnost promovisanja proizvoda ili usluga.

3.1.17. Ocjenjivanje

Pružanje mogućnosti korisnicima da ocjenjivaju bilo šta putem aplikacije postaje uobičajeno. Kako bi podržali ovu funkcionalnost, BaaS platforme pružaju programerima osnovni sistem ocjenjivanja kao dio svoje ponude za rangiranje.

3.1.18. Chat komunikacija

Chat koji funkcioniše u realnom vremenu je uobičajena funkcionalnost u mobilnim i web aplikacijama. BaaS platforme teže da obezbijede eksterne biblioteke za chat funkcionalnosti.

3.1.19. Email

Email je prisutan svugdje. Ima smisla da je to funkcionalnost koja se podrazumijeva na svim BaaS platformama. Infrastruktura emaila može biti dugotrajna za postavljanje i održavanje od strane programera. BaaS uslužitelji pružaju svoje ili koriste usluge nekog drugog uslužitelja, ali email je definitivno prisutan na skoro svim platformama.

3.1.20. Drugi sistemi za razmjenu poruka

Sa emailom, SMS-om i chatom ugrađenim u BaaS sisteme, neki uslužitelji obezbjeđuju kompletan sistem za razmjenu poruka, te upravljanje istim.

3.1.21. Obavijesti (engl. *push notifikacije*)

Push obavijesti su uobičajene na pametnim telefonima. Korisnici očekuju da će im informacije i poruke biti poslane. Push obavijesti su daleko najčešći alat za komunikaciju i razmjenu poruka koji BaaS uslužitelji pružaju programerima.

3.1.22. SMS

Budući da je navedeno da se BaaS još može nazvati i mobilni BaaS (MBaaS), tako i BaaS platforme su više okrenute prema programerima za mobilne aplikacije. U mobilnim aplikacijama SMS je najdominantniji način komunikacije. SMS funkcionalnost je osnova većine BaaS platformi.

3.1.23. Lokacija

Geolokacija pruža programerima izvorni način za identifikaciju lokacije mobilnog telefona, korisnika ili računara povezanog na internet. Lokacija korisnika se obezbjeđuje preko geografske širine / dužine ili adrese lokacije, tako da se funkcionalnost lokacije može koristiti u aplikacijama.

3.1.24. Mjesta

U današnjem svijetu mobilnih aplikacija, programerima su potrebni visokokvalitetni podaci o poslovanju i drugim mjestima od interesa. BaaS teži da pruži podatke o izvornim mjestima, obično od uslužitelja treće strane.

3.1.25. Ciljanje

Geo ciljanje (engl. *Geo Targeting*) korisnika je općenito ciljanje korisnika na temelju nekoliko tačaka podataka, uključujući lokaciju, ponašanje, povijest, prijatelje i druge vrijedne informacije. Ciljanje je nešto na šta bi developeri morali da potroše mnogo vremena na razvoj, ali BaaS platforme to već pružaju.

3.1.26. Igre

Ospozljavanje programera za izgradnju aplikacija za web i mobilne igre je uobičajeni fokus za BaaS uslužitelje. Igre su daleko najveći fokus BaaS uslužitelja, dok ostali podržavaju generički pristup razvoju aplikacija bez konkretnog fokusa.

3.1.27. Dostupnost i skaliranje

Sažetak kompleksnosti backend infrastrukture je jedno od osnovnih načela BaaS-a. Obezbeđujući platformu koja se automatski prilagođava programerima, te ono što im je potrebno je broj jedan razlog zbog kog će programeri koristiti BaaS.

3.1.28. Android

Android je platforma broj dva za mobilne programere. Većina BaaS uslužitelja podržava razvoj aplikacija Android aplikacija.

3.1.29. Blackberry

Blackberry je još uvijek popularan uređaj u kompanijama i na nekim tržištima širom svijeta. Neki BaaS uslužitelji isporučuju Blackberry SDK-ove i alate za programere koji žele da razvijaju aplikacije za ove uređaje.

3.1.30. iOS

iOS je platforma broj jedan za koju programeri prave aplikacije. Svi BaaS uslužitelji nude alate za implementaciju mobilnih aplikacija na iOS platformama koja cilja na iPhone ili iPad uređaje.

3.1.31. HTML5

HTML5 (engl. *Hyper Text Markup Language*) je jezik za strukturiranje i prezentiranje sadržaja za World Wide Web i jezgrenu tehnologiju interneta. HTML5 je postao standard za razvoj mobilnih aplikacija, bile one na webu, ili ne. Mnoge BaaS platforme nude HTML5 kao alternativu izvornim iOS ili Android aplikacijama za programere.

3.1.32. Podrška za kompanije

Mnogi BaaS uslužitelji imaju već gotova rješenja vezana za kompanije, uključujući prodaju i podršku. BaaS platforme smatraju, te iskorištavaju sve potencijale za izradu rješenja za kompanije, a kao dio njihovih poslovnih strategija.

4. PREDLOŽENO PROGRAMSKO RJEŠENJE – BAAS SISTEM KROZ PROGRAMSKI JEZIK PYTHON

U ovom poglavlju priloženo je programsko rješenje koristeći programski jezik Python, a koje služi za jednostavnu pohranu podataka. Rješenje implementira REST API koji je objašnjen u prethodnom poglavlju, a koristeći uobičajene biblioteke i proširenja koji se koriste prilikom implementacije ovakvih tipova aplikacija i programa.

4.1. Opis korištenih tehnologija

U nastavku je dat opis korištenih tehnologija i proširenja koje su se koristile prilikom razvoja ovog programskog rješenja.

4.1.1. Programska jezik Python

Python je jedan od najpopularnijih programskih jezika današnjice. To je jezik opće namjene, interpretiran i visoke razine. Kreiran je od strane Guida van Rossuma i prvi put je predstavljen 1991. godine. Filozofija ovog jezika ogleda se u čitljivosti, te uveliko doprinosi organizovanosti malih i velikih projekata.

Python podržava više programskih paradigma uključujući proceduralno, funkcionalno, te objektno-orientisano programiranje, a uz to je dinamički programska jezik i posjeduje garbage collector koji je zadužen za upravljanje memorijom.

Namjenjen je da bude čitljiv programski jezik, te radi na principu uvlačenja redova za razgraničavanje određenih blokova koda, za razliku od većine programskih jezika koji koriste vitičaste zgrade. Tačka-zarez “;“ simbol nije obavezan na kraju reda. Python posjeduje duck typing pristup, odnosno, tipovi postoje, ali nisu obavezni da se navode prilikom inicijalizacije varijabli.

Standardna biblioteka Python-a pruža mnogo alata za razne zadatke. Za spajanje na internet koristi mnoge formate i protokole, a jedni od njih koji su podržani su HTTP i MIME. Uključuje module za kreiranje grafičkih interfejsa za korisnike, povezivanje sa bazama podataka, unit testiranje itd. PyPI (engl. Python Package Index) je zvanično skladište biblioteka za Python, te sadrži preko 130 hiljada paketa i biblioteka sa širokim spektrom funkcionalnosti, kao što su:

- ❖ Grafički korisnički interfejsi
- ❖ Web okviri
- ❖ Multimedija
- ❖ Baze podataka
- ❖ Umrežavanje
- ❖ Okviri za testiranje
- ❖ Automatizacija
- ❖ Web scraping
- ❖ Alati za dokumentacije projekata
- ❖ Administracija sistema
- ❖ Obrada teksta
- ❖ Obrada slike¹¹

¹¹ <https://www.python.org/> (20.06.2019.)

4.1.2. PyCharm

PyCharm je integrisano razvojno okruženje (engl. *integrated development environment - IDE*) i koristi se za kompjutersko programiranje, specifično za Python programski jezik. Razvijen je od strane češke kompanije JetBrains, a podržava analitiku koda, grafički debugger, integrisane alate za unit testiranje, integraciju sa alatima za organizaciju i spremanje koda, te podržava web razvoj zajedno za Django tehnologijama.

PyCharm posjeduje verzije za Windows, macOS i Linux operativne sisteme, a postoje i besplatne, te profesionalne verzije sa dodatnim funkcionalnostima, a koje se mogu dobiti kupovinom licence za korištenje.¹²

4.1.3. Flask

Flask je mikro web okruženje napisano u Python programskom jeziku. Klasificirano je kao mikro okruženje zato što ne zahtjeva dodatne alate i pakete, te može raditi neovisno. Pogodan je za početnike pošto zahtjeva veoma nizak nivo konfiguracije za pokretanje jednostavnih aplikacija.

Flask je proizведен 2010. godine od strane Armina Ronachera kao alternativa na Django okruženje koje je tada bilo jedino dostupno za korištenje. Na samom početku je postojalo mnogo problema prilikom organizacije koda, a najviše problema je bilo vezano za prilikom spremanja i povlačenja koda koji koristi Flask. Kasnije je Ronacher stvorio posebno skladište za Flask kod, te uklonio određene nedostatke, a danas to skladište posjeduje i nove razne biblioteke i pakete kao što su Lektor, Jinja i mnogi drugi.

Osim Flask-a, u programskom rješenju koristi se **Flask SQLAlchemy**, te **Flask Restful**. SQLAlchemy je proširenje za Flask, a koristi se za pristup SQL bazama podataka, te pruža razne funkcionalnosti koje pomažu programerima prilikom povezivanja aplikacija na bazu podataka. Restful je takođe proširenje za Flask koji podržava kreiranje REST API-a, te je iznimno jednostavan za upotrebu i samu integraciju u programsко rješenje.¹³

4.1.4. SQLite

SQLite je sistem za upravljanje relacionim bazama podataka (engl. *relational database management system - RDBMS*). Za razliku od drugih sistema za upravljanje, SQLite nije klijent – server baza podataka, nego je ugrađena u sami backend sistem. Implementira većinu SQL standarda koristeći PostgreSQL sintaksu. Međutim, SQLite koristi dinamičke tipove, što znači da, na primjer, u kolonu tekstualnog tipa je moguće spremiti niz cijelih brojeva. SQLite će pokušati uraditi konverziju tipa, ali ako ne uspije, podatke će spremiti u obliku koji je čitljiv samoj bazi podataka.

SQLite je prilično popularan izbor kao već ugrađeni softver za baze podataka na klijentskoj strani, a koristi se za skladištenje podataka kao što su web pretraživači.

¹² <https://www.jetbrains.com/pycharm/features/> (22.06.2019.)

¹³ <http://flask.pocoo.org/> (23.06.2019)

Trenutno je najrasprostranjeniji mehanizam baze podataka za web pretraživače, operativne sisteme i već ugrađenih sistema kao što je to slučaj na mobilnim telefonima.¹⁴

4.1.5. JavaScript programski jezik

JavaScript (skr. *JS*) je interpretirani programski jezik visoke razine, koji je u skladu sa ECMAScript specifikacijama. Ima sintaksu baziranu na vitičastim zgradama, dinamičko tipovanje, objektnu-orientaciju zasnovanu na prototipu, te funkcije prve klase.

Pored HTML-a i CSS-a, JavaScript je jedna od ključnih tehnologija World Wide Web-a. Omogućava interaktivne web stranice i veoma je bitan dio web aplikacija. Skoro sve web stranice ga koriste, a glavni web preglednici imaju namjenski JavaScript mehanizam za njegovo izvršavanje.

Na samom početku, JavaScript je zamišljen samo kao jezik koji se izvršava na strani klijenta u web preglednicima, ali danas su JavaScript mehanizmi ugrađeni u mnoge tipove serverskih softvera, a osim toga i u aplikacijama i programima koje nisu web bazirane, kao što su procesori teksta, PDF softveri, te u mobilnim i desktop aplikacijama.¹⁵¹⁶

4.1.6. JSON format podataka

JSON (engl. JavaScript Object Notation) je lagani format za razmjenu podataka. Ljudima je lako čitati i pisati ovaj format, te je jednostavan za analiziranje i generisanje od strane računara. Zasnovan je na objektima iz JavaScript programskog jezika, standard ECMA-262 3RD Edition – decembar 1999. JSON je tekstualni format koji je potpuno neovisan o jeziku, ali koristi konvencije koje su poznate programerima porodice C-jezika, uključujući C, C++, C#, Java, JavaScript, Perl, Python, ali i mnoge druge. Ova svojstva čine JSON idealnim načinom za razmjenu podataka.¹⁷

```
{  
  "userId": 1,  
  "id": 1,  
  "title": "delectus aut autem",  
  "completed": false  
}
```

Slika 3. Primjer jednog JSON objekta

4.1.7. Heroku platforma

Heroku je PaaS platforma u oblaku koja podržava više programskih jezika. Heroku je jedna od prvih platformi u oblaku od 2007. godine kada je podržavala jedino programski jezik Ruby. Danas podržava Java, Node.js, Scala, Clojure, Python, PHP i GO. Naziva se još poliglotskom platformom budući da podržava priličan broj programskih jezika i

¹⁴ <https://www.sqlite.org/index.html> (23.06.2019.)

¹⁵ <https://en.wikipedia.org/wiki/JavaScript> (25.06.2019.)

¹⁶ <https://www.javascript.com/> (26.06.2019.)

¹⁷ <https://www.json.org/> (26.06.2019.)

tehnologija. Sadrži dosta funkcionalnosti koje programerima omogućavaju da raspoređivaju, pokrenu, te skaliraju svoje aplikacije.¹⁸

4.2. Implementacija

U ovom poglavlju definisani su određeni koraci da bi se implementirao jednostavan BaaS sistem, koji uključuje REST API, SDK, te klijentsku web aplikaciju koja koristi SDK da bi se povezala sa REST API-jem.

4.2.1. Implementacija REST API-ja u programskom jeziku Python

Kao prvi korak prilikom implementacije jednostavnog REST API-ja u programskom jeziku Python kreirat ćemo novu fasciklu u kojoj će biti smještena naša aplikacija. Možemo je nazvati **python_rest**.

❖ Baza podataka

Kao što smo već naveli, u programskom rješenju koristiti će se SQLite baza podataka. Kao prvi korak moramo preuzeti instalacijske datoteke za SQLite bazu podataka za određeni operativni sistem preko sljedećeg linka: <https://www.sqlite.org/download.html> .

Nakon što smo uspješno preuzezeli i instalirali SQLite bazu podataka, naredni korak je da kreiramo istu koju će da koristi REST API aplikacija. To možemo da uradimo kroz komandni prozor, te promjenimo navigaciju na novokreiranu fasciklu **python_rest**. Nakon toga izvršimo sljedeću komandu:

```
python_rest > $ sqlite3 baas.db
```

Ukoliko je baza podataka uspješno kreirana, naredni korak je da se kreiraju odgovarajuće tabele. U ovoj prezentaciji biće kreirana jedna tabela koja će biti zadužena za čuvanje podataka, na primjer studenata. U tom slučaju tabela će da sadrži jedinstveni identifikator (ID) koji će istovremeno da služi i kao primarni ključ, te ime i prezime studenta (name). To možemo uraditi sa sljedećom komandom u istom prozoru:

```
> CREATE TABLE student(
    id integer primary key autoincrement,
    name varchar
);
```

Ukoliko želimo da provjerimo da li je baza podataka, ali i tabela uspješno kreirana možemo da kreiramo nekoliko zapisa u tabelu student. To možemo da uradimo sa sljedećim komandama:

```
> INSERT INTO student(name) VALUES ('Student 1');
> INSERT INTO student(name) VALUES ('Student 2');
```

¹⁸ <https://www.heroku.com/> (27.06.2019.)

```
> INSERT INTO student(name) VALUES ('Student 3');  
> INSERT INTO student(name) VALUES ('Student 4');  
> INSERT INTO student(name) VALUES ('Student 5');  
> INSERT INTO student(name) VALUES ('Student 6');  
> SELECT * FROM student;
```

Nakon što su izvršene ove komande, ispis na ekranu bi trebao da bude sljedeći:

```
1|student1  
2|student2  
3|student3  
4|student4  
5|student5  
6|student6
```

To znači da smo uspješno unijeli podatke u našu bazu podataka, te ukoliko vidimo datoteku sa nazivom **baas.db** unutar fascikle **python_rest** to znači da je ovaj korak uspješno završen i da se može preći na sljedeći.

❖ REST API

Budući da je baza podataka uspješno kreirana možemo početi sa implementacijom REST API-ja. Prvi korak je preuzimanje i instalacija Python-a, a potrebne instalacijske datoteke se mogu pronaći na linku <https://www.python.org/downloads/>. Osim toga, potrebno je preuzeti PIP instalacijsku datoteku na <https://bootstrap.pypa.io/get-pip.py>. PIP je alat koji pristupa magacinu biblioteka i paketa za Python okruženje, te radi preuzimanje i potrebnu instalaciju. Postavka ovog alata zahtjeva prethodnu instalaciju Python-a, a izvršava se tako što se otvori komandni prozor, navigira do fascikle u kojoj se nalazi get-pip.py datoteka, te se izvrši sljedeća komanda.

```
downloads > $ python get-pip.py
```

Nakon što smo preuzeли i instalirali potrebne stvari za rad aplikacije, naredni korak je kreirati u **python_rest** fascikli **server.py** datoteku u kojoj će biti smješten kod za interakciju sa bazom podataka, te definicije ruta za pristup API-ju. Pošto se radi o demonstraciji dovoljna je samo ova datoteka, međutim za projekte većeg obima radi se o mnogo više datoteka, raznih namjena i funkcionalnosti da bi se bolje organizovao kod.

Pošto je kreirana datoteka za kod, pristupa se inicijalizaciji projekta, odnosno preuzimanju i integrisanju potrebnih paketa za nesmetan rad aplikacije. Za to je potrebno da ponovo otvorimo komandni prozor te navigiramo do foldera **python_rest**. Nakon toga izvršimo sljedeći set komandi:

```
python_rest > $ pip install virtualenv  
python_rest > $ virtualenv venv
```

```
python_rest > $ source venv/bin/activate
python_rest > $ pip install flask flask-jsonpify flask-
sqlalchemy flask-restful
```

Prethodno navedeni set komandi radi sljedeće: preuzimanje i instaliranje **virtualenv** biblioteke (zadužene za kreiranje izoliranog okruženja za Python), postavljanje izoliranog okruženja, aktiviranje istog, instalacija potrebnih biblioteka uključujući flask.

Konačno možemo da pređemo na implementaciju REST API-ja, odnosno pisanje koda u server.py datoteku. Nakon što otvorimo našu datoteku kroz PyCharm razvojno okruženje, prvo možemo da definišemo zaglavlje, a to se odnosi na importovanje svih potrebnih prethodno instaliranih biblioteka.

```
from flask import Flask, request
from flask_restful import Resource, Api
from sqlalchemy import create_engine
from json import dumps
from flask.ext.jsonpify import jsonify
```

Naredni korak je instanciranje aplikacije, odnosno varijabli za bazu podataka i API. Nakon zaglavlja pišemo sljedeći kod koji pravi instancu baze podataka, aplikacije, te API-ja.

```
db_connect = create_engine('sqlite:///baas.db')
app = Flask(__name__)
api = Api(app)
```

Nakon definicije potrebnih varijabli pristupa se implementaciji funkcija koje su zadužene za čitanje podataka iz baze podataka. U ovom slučaju kreirane su dvije funkcije, i to za čitanje svih zapisa, te jednog zapisa koji se čita prema datom identifikacijskom broju.

```
class Students(Resource):
    def get(self):
        conn = db_connect.connect()
        query = conn.execute("select * from student;")
        result = {'students': [query.cursor.fetchall()]}
        return jsonify(result)

class Student(Resource):
    def get(self, student_id):
        conn = db_connect.connect()
        query = conn.execute("select * from student where id =%d " %int(student_id))
        result = {'data': [dict(zip(tuple (query.keys()), i)) for i in query.cursor]}
        return jsonify(result)
```

Kao što vidimo iz prethodno definisanog koda, kreirane su dvije klase Students i Student koje posjeduju po jednu GET metodu, a koje su zadužene, respektivno, za čitanje svih zapisa iz tabele student i za čitanje jednog zapisa iz tabele student. Osim toga, vidimo da je prvi korak uspostavljanje konekcije sa bazom podataka, definisanje upita, te izvršavanje istog. Zadnja linija vraća željeni rezultat iz funkcije.

Pošto su definisane dvije funkcije, prelazimo na definisanje API-ja za pristup tim metodama preko HTTP protokola. To radimo sa sljedećim kodom koji postavljamo neposredno ispod prethodno definisanih metoda:

```
api.add_resource(Students, '/students')
api.add_resource(Student, '/student/<student_id>')
```

Ove dvije linije koda dodaju resurse u sljedećem formatu: naziv klase, ruta kao string tip. U ovom slučaju definisane su dvije rute, jedna za klasu Students, a druga za klasu Student.

Na kraju, neophodno je definisati kod za pokretanje aplikacije:

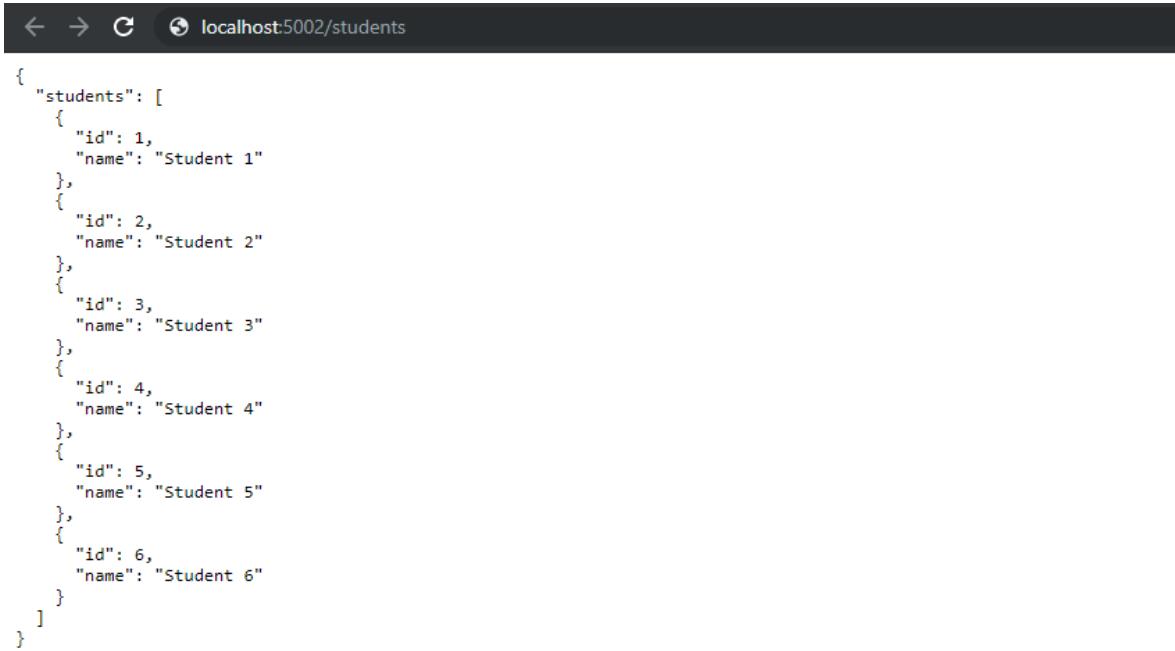
```
if __name__ == '__main__':
    app.run(port='5002')
```

Primjetit ćemo da u dvije prethodno definisane dvije linije koda imamo uslov koji ispitiva naziv aplikacije, a koji će u našem slučaju biti zadovoljen, budući da je osnovni, odnosno ugrađeni naziv aplikacije '`__main__`', ukoliko to nije drugačije definisano konfiguracijskom datotekom. Ukoliko je uslov zadovoljen, aplikacija se pokreće, a u ovom slučaju na portu 5002.

Zaista, ukoliko pokrenemo aplikaciju sa komandom:

```
python_rest > $python server.py
```

te otvorimo web preglednik i nakon što unesemo sljedeći URL:
<http://localhost:5002/students>, možemo vidjeti sljedeći ispis na ekranu:

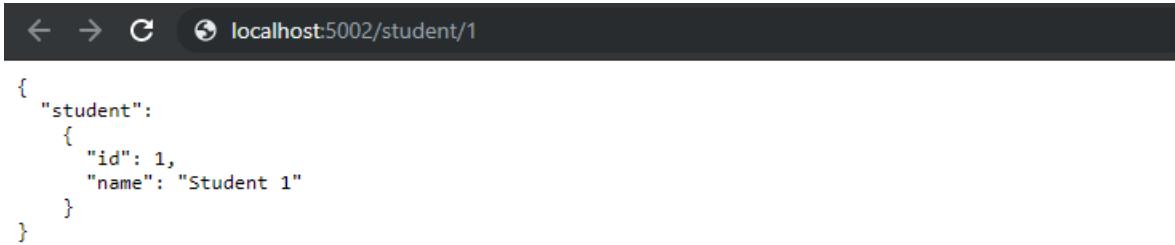


```
{ "students": [ { "id": 1, "name": "Student 1" }, { "id": 2, "name": "Student 2" }, { "id": 3, "name": "Student 3" }, { "id": 4, "name": "Student 4" }, { "id": 5, "name": "Student 5" }, { "id": 6, "name": "Student 6" } ] }
```

Slika 4. Ispisa svih zapisa iz tabele student u JSON formatu

Sa prethodne slike vidimo da smo dobili ispis svih zapisa iz tabele **student** u JSON formatu.

Isto tako, ukoliko unesemo sljedeći URL: <http://localhost:5002/student/1>, možemo vidjeti ispis jednog zapisa iz tabele **student** sa ID-em 1:



```
{ "student": { "id": 1, "name": "Student 1" } }
```

Slika 5. Ispis jednog zapisa iz tabele student sa ID-em 1 u JSON formatu

Budući da smo dobili željene rezultate, time je implementacija REST API-ja završena, te se može preći na sljedeći korak, a to je raspoređivanje na jedan od servisa.

❖ Raspoređivanje REST API-ja na Heroku platformu

Prije nego pristupimo samoj operaciji raspoređivanja (engl. *deployment*) moramo da pripremimo projekat za raspoređivanje. Sama priprema za raspoređivanje je drugačija u zavisnosti od platforme, odnosno servisa na koju se raspoređuje. U ovom slučaju REST API ćemo rasporediti na Heroku platformu, stoga ćemo definisati neophodne korake koji su potrebni da bi aplikacija funkcionsala nakon samog raspoređivanja.

Prvi korak je kreiranje nove datoteke pod nazivom **requirements.txt** u **python_rest** fascikli. Nakon što smo uspješno kreirali datoteku, otvaramo komandni prozor unutar naše glavne fascikle. Komanda koju treba da izvršimo je:

```
python_rest > $ pip freeze
```

Prethodna komanda bi trebala da ispiše sljedeće:

```
aniso8601==7.0.0
click==6.7
Flask==0.12.2
Flask-Cors==3.0.3
Flask-Jsonpify==1.5.0
Flask-RESTful==0.3.7
itsdangerous==0.24
Jinja2==2.10
MarkupSafe==1.0
pyodbc==4.0.22
pytz==2017.3
six==1.11.0
SQLAlchemy==1.3.5
virtualenv==16.6.1
Werkzeug==0.14.1
```

Prethnoodna lista ispisuje sve pakete koji su instalirani za ovu Python aplikaciju, odnosno sve ono što smo unosili nakon **pip install** komande, pa čak i dodatne biblioteke koje su automatski instalirane zato što neki paketi to zahtjevaju. Osim naziva paketa vidimo i verzije istih. U svakom slučaju, ukoliko smo uspješno ispisali pakete, kopiramo taj ispis, te zalijepimo u **requirements.txt** datoteku i spremimo. Ova datoteka, kada se rasporedi na Heroku platformu zajedno za kodom, govori operativnom sistemu koji su paketi neophodni da bi aplikacija uspješno radila, te platforma, u ovom slučaju Heroku, uspješno instalira sve potrebne pakete i biblioteke. Ova datoteka je neophodna za sve platforme prilikom raspoređivanja Python aplikacije.

Naredni korak je kreiranje datoteke koja je specifična za Heroku. Naziv treba da bude **Procfile** (bez ekstenzije, odnosno tipa datoteke). Ova datoteka treba da sadrži sljedeće:

```
web: python server.py $PORT
```

Format ove datoteke za Python aplikacije je specifično zadan od strane Heroku-a, te možemo primjetiti da smo definisali naziv naše glavne datoteke gdje je smješten kod (server.py), te varijabla \$PORT koja, u ovom slučaju, će preuzeti vrijednost koju operativni sistem automatski dodjeli, ali i korisnik je u mogućnosti da definiše broj porta. Međutim, u tom slučaju nije zagarantovano da će aplikacija biti uspješno pokrenuta (engl.

build and run), a u zavisnosti od raspoloživosti tog porta.

Zadnja datoteka koju moramo da kreiramo nosi naziv **runtime.txt**, te treba da sadrži sljedeće:

```
python-3.6.2
```

Dakle, definicija tipa aplikacije uz verziju. Treba napomenuti da se ne može dodijeliti bilo koja verzija, već **3.6.2, 3.6.6 i 3.7.0** prema zvaničnoj Heroku dokumentaciji. Ono što je bitno, verzija Python-a koji je instaliran na našem sistemu ne mora nužno da se podudara sa gore navedenim.

Nakon što smo kreirali sve potrebne datoteke, naredni korak je kreiranje računa na www.heroku.com. Kada korisnik kreira, te verifikuje račun otvara se početna stranica koja prikazuje listu kreiranih aplikacija na ovoj platformi. U gornjem desnom uglu stranice nalazi se dugme „New“. Klikom na to dugme odaberemo „Create new app“, a nakon toga otvara se forma za unos podataka o novoj aplikaciji.

The screenshot shows the 'Create New App' interface. At the top right is a 'Create New App' button. Below it is an 'App name' input field containing 'app-name'. Underneath is a 'Choose a region' dropdown menu set to 'United States'. At the bottom left is a 'Create app' button.

Slika 5. Forma za kreiranje aplikacije na platformi Heroku

U polje „App name“ unosimo željeni naziv aplikacije. Naziv treba da bude smislen, a isti će biti sadržan i u URL-u za pristup našoj aplikaciji. Osim naziva, možemo da odaberemo region servera na kojem će da bude smještena naša aplikacija, a imamo dva izbora, United States (SAD) i Europa. U našem slučaju, definisali smo naziv aplikacije „baas-backend“ i kao region odabrali Europu. Klikom na dugme „Create app“ aplikacija je kreirana, te je korisnik preusmjeren na novi prozor koji je ustvari kontrolna ploča za našu aplikaciju, te je

unaprijed odabran „Deploy“ tab, a to je ono što nam treba u ovom trenutku. Na tom prozoru možemo da vidimo korake za raspoređivanje aplikacije, a kao prvi korak je odabir metode za raspoređivanje.



Slika 6. Moguće metode za raspoređivanje aplikacije na Heroku

Naš odabir je Heroku Git, odnosno prva opcija koja je unaprijed odabrana. U nastavku možemo da vidimo naredne korake koje je potrebno poduzeti da bi se aplikacija uspješno rasporedila.

The screenshot shows the Heroku Git deployment steps:

- Deploy using Heroku Git**: Use git in the command line or a GUI tool to deploy this app.
- Install the Heroku CLI**: Download and install the [Heroku CLI](#). If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.
Command: \$ heroku login
- Create a new Git repository**: Initialize a git repository in a new or existing directory.
Commands: \$ cd my-project/
\$ git init
\$ heroku git:remote -a baas-backend
- Deploy your application**: Commit your code to the repository and deploy it to Heroku using Git.
Commands: \$ git add .
\$ git commit -am "make it better"
\$ git push heroku master
- Existing Git repository**: For existing repositories, simply add the `heroku` remote.
Command: \$ heroku git:remote -a baas-backend

Slika 7. Koraci za raspoređivanje aplikacije preko Heroku Git opcije

Dakle, kao prvi korak moramo da preuzmemo i instaliramo Heroku CLI na sljedećem linku: <https://devcenter.heroku.com/articles/heroku-cli#download-and-install>. Nakon što smo uspješno preuzeeli i instalirali Heroku CLI, odnosno alat za raspoređivanje aplikacije na Heroku platformu, otvorimo komandni prozor u glavnoj fascikli te unesemo:

```
python_rest > $ heroku login
```

Nakon unosa ove komande na zahtjev unosimo korisničko ime, odnosno email i lozinku koju smo definisali prilikom registracije. Nakon prijave preko komandne linije treba da inicijaliziramo GIT. To radimo sa sljedeće dvije komande:

```
python_rest > $ git init
python_rest > $ heroku git:remote -a baas-backend
```

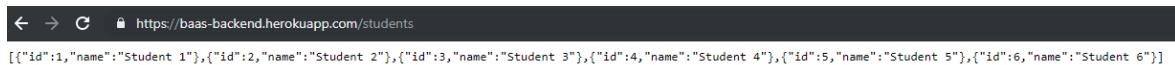
Prva komanda inicijalizuje GIT spremište (engl. *repository*), a druga definiše spremište u koje ćemo postaviti naš kod iz aplikacije.

Nakon uspješne GIT inicijalizacije, naredni korak je postavljanje i raspoređivanje aplikacije. To radimo sa sljedećim koracima:

```
python_rest > $ git add .
python_rest > $ git commit am "initial commit"
python_rest > $ git push heroku master
```

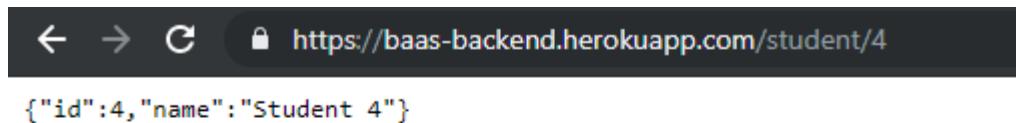
Prva komanda iz ovog seta komandi dodaje sve izmijenjene, u našem slučaju sve datoteke i fascikle iz naše aplikacije u prvi nivo spremišta. Naredna dodaje u drugu fazu uz određenu poruku (initial commit), te konačno treća komanda dodaje kod na udaljeno Heroku spremište, te automatski radi raspoređivanje.

Na kraju, ukoliko je raspoređivanje uspješno izvršeno možemo da provjerimo naše definisane krajnje tačke (engl. *endpoints*) tako što ćemo kliknuti na dugme „Open app“ u gornjem desnom uglu stranice. Nakon toga, na URL dodamo naše definisane endpointe, a to su **/students** i **students/:id**.



Slika 8. Endpoint koji ispisuje sve studente

Sa prethodne slike vidimo da ovaj endpoint je uspješno ispisao svih 6 studenata iz naše baze podataka.



Slika 9. Endpoint koji ispisuje studenta sa ID-em 4

Takođe, vidimo da endpoint koji ispisuje jednog studenta, u ovom slučaju sa ID-em 4 radi savršeno svoj posao.

Nakon što smo uspješno rasporedili REST API na Heroku platformu možemo da pristupimo izradi jednostavnog SDK-a.

4.2.2. Implementacija SDK-a

Budući da želimo da povežemo implementirani backend, odnosno REST API sa nekom web aplikacijom, u nastavku će biti prezentovana implementacija osnovnog SDK-a za JavaScript programski jezik.

SDK će da sadrži jednu JavaScript datoteku koja će imati svega dvije funkcije, odnosno po jedna za svaki API sa backenda. Prvi korak bi bio kreiranje JavaScript datoteke unutar **python_rest** fascikle, pod nazivom **student-sdk.js**. Nakon što je kreirana datoteka, možemo pristupiti implementaciji.

Nakon što smo otvorili novokreiranu, odnosno praznu JavaScript datoteku u preferiranom editoru, definisat ćemo prvi blok koji je ustvari varijabla pod nazivom StudentSDK.

```
var StudentSDK = {  
}
```

Prva stvar koju ćemo da uradimo je da definišemo bazni URL, odnosno URL od naše Heroku aplikacije.

```
var StudentSDK = {  
    baseUrl: ' https://baas-backend.herokuapp.com'  
}
```

Dalje, unutar definisane varijable implementirati ćemo dvije navedene funkcije.

Prva funkcija će biti za dobavljanje svih studenata, te njena implementacija bi trebala da izgleda sljedeće:

```
var StudentSDK = {  
    baseUrl: ' https://baas-backend.herokuapp.com',  
    getStudents: function(callback) {  
        var xhr = new XMLHttpRequest();  
        xhr.onreadystatechange = function() {  
            if (xhr.readyState === 4) {  
                if(callback){  
                    callback(xhr.responseText);  
                }  
            }  
        }  
    }  
}
```

```

        } ;
        xhr.open('GET', StudentSDK.baseUrl +
'/students') ;
        xhr.send() ;
    }
}

```

Dakle, funkcija **getStudents** od ulaznih parametara prima varijablu naziva **callback**, a to je ustvari funkcija kojoj će biti proslijeđena vrijednost, odnosno JSON objekat nakon komunikacije sa REST API-jem. Nakon toga, na samom početku funkcije inicijaliziran je objekat **XMLHttpRequest** koji posjeduje sve funkcije i metode za komunikaciju sa određenim REST API-jem. Funkcija **onreadystatechange** je zadužena da primi željeni rezultat, te procesira ga dalje, a u ovom slučaju da proslijedi u funkciju **callback**. Osim toga, pozvane su dvije funkcije, a to su za definisanje tipa metode i URL-a sa kojim se komunicira, te za konačnu uspostavu veze između SDK-a i REST API-ja.

Slično, funkcija **getStudent** za dobavljanje jednog studenta prema ID-u je definisana na sljedeći način:

```

var StudentSDK = {
    baseUrl: ' https://baas-backend.herokuapp.com',
    getStudents: function(callback) {
        var xhr = new XMLHttpRequest();
        xhr.onreadystatechange = function() {
            if (xhr.readyState === 4) {
                if(callback) {
                    callback(xhr.responseText);
                }
            }
        };
        xhr.open('GET', StudentSDK.baseUrl +
'/students') ;
        xhr.send() ;
    },
    getStudent: function(id, callback) {
        var xhr = new XMLHttpRequest();
        xhr.onreadystatechange = function() {
            if (xhr.readyState === 4) {
                if(callback) {
                    callback(xhr.responseText);
                }
            }
        };
        xhr.open('GET', StudentSDK.baseUrl + '/student/' +
+ id) ;
        xhr.send() ;
    }
}

```

Vidimo se da funkcija **getStudent** razlikuje od funkcije **getStudents** po parametrima. Odnosno, nova funkcija, osim funkcije **callback**, još prima i **id** varijablu koja je analogna ID-u studenta. Ostatak implementacije je isti osim definisanja URL-a za dobavljanje jednog studenta što se može primjetiti iz prethodnog predloška.

Nakon što smo implementirali JavaScript datoteku, te istu spremili možemo je uploadovati na neki od besplatnih servisa, da bi bila dostupna. U ovom slučaju, na hosting YourJavaScript.com na sljedećem linku <http://yourjavascript.com/143493759/student-sdk.js> (07.03.2019.).

Budući da je završen SDK za REST API, možemo pristupiti implementaciji jednostavne klijentske web aplikacije, odnosno da bi povezali SDK sa web aplikacijom te prezentovali ponašanje.

4.2.3. Implementacija klijentske web aplikacije

U ovom koraku biće predstavljeni osnovni koraci za implementaciju jednostavne web aplikacije, te njeno povezivanje sa implementiranim REST API-jem preko definisanog SDK-a.

Implementaciju počinjemo tako što kreiramo HTML datoteku, koja će ujedno i biti jedina u našoj web aplikaciji. Definišemo naziv datoteke kao index.html, te istu otvorimo u preferiranom editoru. Prvi korak je definisanje osnovih HTML elemenata.

```
<!DOCTYPE html>
<html>

<head>
</head>

<body>

</body>

</html>
```

Nakon toga treba da definišemo skripte, odnosno CSS datoteke koje želimo da koristimo. U našem slučaju definisat ćemo putanju do SDK-a na domeni YourJavaScript.com, te Bootstrap.css datoteku koja će nam pomoći da definišemo vizuelno ljepši korisnički doživljaj. Bootstrap je besplatno CSS okruženje koje ćemo da iskoristimo. To radimo na sljedeći način:

```
<head>
    <script type="text/javascript"
src="http://yourjavascript.com/143493759/student-
```

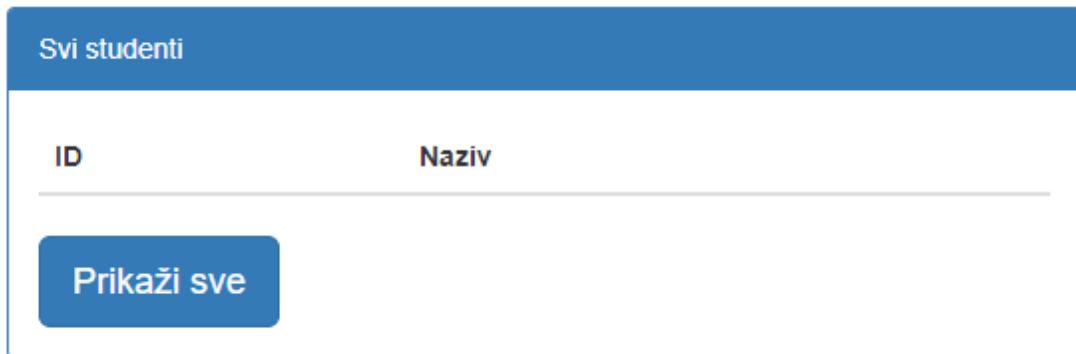
```
sdk.js"></script>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/boostrap.min.css">
</head>
```

Prelazimo na dizajn naše stranice, te definišemo dva panela od kojih će jedan prikazivati listu svih studenata, a drugi jednog studenta u zavisnosti koji ID korisnik unese.

```
<div class="panel panel-primary">
    <div class="panel-heading">Svi studenti</div>
    <div class="panel-body">
        <table id="student_list" class="table table-striped" >
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Naziv</th>
                </tr>
            </thead>
            <tbody></tbody>
        </table>
        <button type="button" id="show_all" class="btn btn-lg
btn-primary" onclick="showAll ()">Prikaži sve</button>
    </div>
</div>
```

Iz prethodnog HTML koda vidimo da smo definisali praznu tabelu sa zaglavljem sačinjenim od polja „ID“ i „Naziv“, te dugme za pokretanje akcije za prikazivanje svih studenata koji posjeduje **onclick** atribut, odnosno definiciju funkcije koja treba da se izvrši klikom na to dugme. Osim toga, vidimo da su ID-evi HTML elemenata već definisani, te CSS klase koje su već definisane u Bootstrap-u.

Ovaj HTML blok u web pregledniku izgleda kao na sljedećoj slici.



Slika 10. Izgled panela za prikaz svih studenata

Nakon toga, prelazimo na implementaciju drugog panela za prikaz jednog studenta. HTML kod je sljedeći:

```
<div class="panel panel-primary">
  <div class="panel-heading">Jedan student</div>
  <div class="panel-body">
    <label><strong>ID: </strong><span
      id="student_id"></span></label><br/>
    <label><strong>Naziv: </strong><span
      id="student_name"></span></label><br/>
    <br/><br/>
    <label>Unesite ID: </label>
    <input type="number" class="form-control"
      id="student_id_input" min="0" />
    <button type="button" id="show_one" class="btn btn-lg
      btn-primary" onclick="showOne () ">Prikaži</button>
  </div>
</div>
</div>
```

Iz prethodnog HTML predloška vidimo da smo u ovom slučaju definisali dvije labele koje prikazuju ID i naziv željenog studenta, polje za unos broja, odnosno ID-a, te dugmeta sa definisanom akcijom za prikazivanje rezultata.

Ovaj panel izgleda kao na sljedećoj slici.

Jedan student

ID:

Naziv:

Unesite ID:

Prikaži

Slika 11. Izgled panela za prikaz jednog studenta

Konačno, možemo da pristupimo implementaciji JavaScript koda koji će obavljati funkcije pritiskom na dugmad, odnosno komunicirati sa REST API-jem preko SDK-a.

Definišemo **script** blok nakon **body** elementa u HTML-u, te definišemo funkciju naziva **showAll**, odnosno istog naziva kao i što smo definisali atribut na dugmetu za prikazivanje svih studenata. Implementacija je sljedeća:

```
function showAll() {
    document.querySelector('#student_list
tbody').innerHTML = '';
    StudentSDK.getStudents(function(students) {
        students = JSON.parse(students);
        var html = '<tbody>';
        for(var i=0; i<students.length; i++) {
            html += '<tr>';
            html += '<td>' + students[i].id + '</td>';
            html += '<td>' + students[i].name + '</td>';
            html += '</tr>';
        }
        html += '</tbody>';
        document.querySelector('#student_list
tbody').innerHTML = html;
    });
}
```

Iz prethodnog JavaScript koda vidimo da prva linija ustvari briše sadržaj iz tabele, te nakon toga se poziva funkcija **getStudents** iz našeg SDK-a. U sklopu poziva te funkcije

definiše se funkcija koja se proslijeđuje u **getStudents** funkciju (engl. *callback*) sa parametrom **students**. Sljedeći korak je konverzija primljenog rezultata **students** u validan JavaScript objekat. Vidimo da je definisana varijabla **html** koja je ustvari HTML predložak koji kreiramo kao sadržaj tabele. Nakon toga je definisana **for** petlja koja prolazi kroz sve elemente niza **students**, te kreira HTML predložak. HTML predložak se na kraju ubacuje u tabelu iz prvog tabela.

Ispis nakon klika na dugme „Prikaži sve“ daje sljedeći rezultat.

Svi studenti	
ID	Naziv
1	Student 1
2	Student 2
3	Student 3
4	Student 4
5	Student 5
6	Student 6

Prikaži sve

Slika 12. Prikaz ispisa svih studenata

Nakon što je prezentovana funkcionalnost prvog panela, možemo preći na definisanje drugog. Ponovo, prvi korak je definisanje funkcije **getOne**, te implementacija iste za prikaz jednog studenta. Implementacija navedene metode je sljedeća.

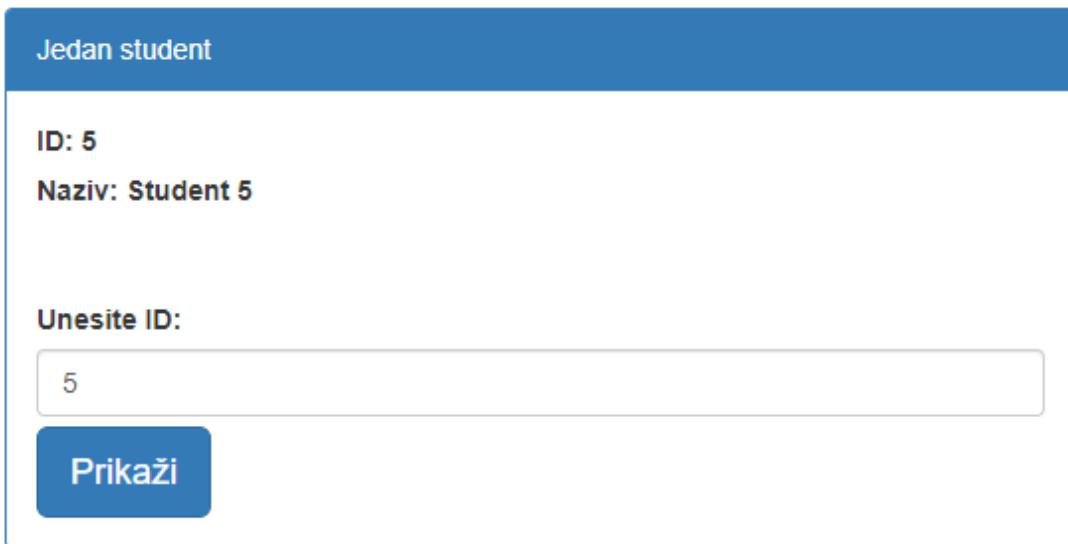
```
function showOne() {
    var studentId = document.getElementById('student_id_input').value;
    if(!studentId) {
        alert('Unesite ID studenta!');
        return;
    }
    document.getElementById('student_id').innerText = '';
    document.getElementById('student_name') =
```

```

    '';
    StudentSDK.getStudent(studentId, function(student) {
        student = JSON.parse(student);
        document.getElementById('student_id').innerText =
student.id;
        document.getElementById('student_name').innerText =
= student.name;
    });
}

```

U prvim linijama ove funkcije uzima se vrijednost iz polja za unos ID-a studenta, te se ista validira. Ukoliko je polje prazno, odnosno bez vrijednosti, korisnik se obavještava, te funkcija završava svoje izvršavanje. Ukoliko je korisnik zaista unio korektnu vrijednost, sadržaj labela za ispis se briše, te se poziva funkcija **getStudent** iz SDK-a, te se proslijedi unijeti ID i definiše povratna funkcija. U povratnoj funkciji se radi konverzija dobijenog rezultata u validan JavaScript objekat, te ispisuju vrijednosti u labelama. Ispis na ovom panelu izgleda kao na sljedećoj slici.



Slika 13. Prikaz ispisa informacija za studenta sa ID-em 5

Sa prethodne slike možemo vidjeti korektan ispis za studenta sa ID-em 5.

Implementacijom klijentske web aplikacije zaokružena je implementacija, odnosno simulacija jednog BaaS sistema koji sadrži sve osnovne značajke, odnosno sistem koji odlikuje ponašanje i komponente BaaS sistema. Naravno, ovo programsko rješenje je sušta prezentacija značajki, te je definisan zbog razumijevanja materije, ali u praksi, naravno ne postoje ovolikо jednostavnvi dijelovi BaaS sistema.

ZAKLJUČAK

Tema ovog rada bila je definisanje pojmove vezanih za BaaS te implementacija jednostavnog BaaS sistema. Tokom analize postojećih BaaS sistema uočeno je mnogo blokova, odnosno jedinki tih sistema koji se nalaze u svakom, a neki od njih su upravljanje korisnicima (registracija, prijavljivanje, pristupi itd.), upravljanje datotekama, pohrana i čitanje podataka. U ovom radu, tokom implementacije definisano je čitanje podataka iz unaprijed definisane baze podataka kroz programski jezik Python. Naravno, postoji prostor da se sistem proširi uz pohranu podataka, dodatnim blokovima i sl.. Svakako, uz drugačije strukturisanje slojeva, a to se podrazumijeva na razdvajanje sloja za pristup bazi podataka, sloj za obradu podataka, te krajnji sloj koji sadrži definicije API-ja, a ukoliko govorimo samo o dijelu REST API-ja. Uz to, moguće je modularno definisanje novih nezavisnih i zavisnih blokova uz isti pristup. Budući da je razvoj ovog dijela BaaS sistema urađen kroz programski jezik Python, potencijal je veliki za buduća proširenja ako se uzmu u obzir mogućnosti tog programskog jezika, te dodatni paketi i biblioteke koje su dostupne da bi se olakšao život programerima koji razvijaju takve tipove sistema.

U drugom dijelu aplikacije razvijen je jednostavni SDK koji služi kao posrednik između klijentskih web aplikacija i razvijenog REST API-ja, a u cilju „prikrivanja“ endpointa i jednostavnije implementacije na klijentskoj strani. U ovom slučaju SDK je razvijen u JavaScript programskom jeziku, ali u praksi razvija se širok spektar SDK-a za razne programske jezike i okruženja. Naravno, i u ovom slučaju postoje mogućnosti za proširenje u skladu sa proširenjima REST API-ja. Budući da je SDK razvijen u JavaScript programskom jeziku za klijentsku stranu, radi demonstracije, implementirana je web aplikacija.

Web aplikacija komunicira sa REST API-jem preko definisanog SDK-a, te ispisuje informacije na ekranu dobavljene od strane REST API-ja. Implementirana aplikacija sadrži dva panela za dvije funkcionalnosti, odnosno dva endpointa za prijem podataka i ispis istih.

Tokom implementacije ovog BaaS sistema, te analize utrošenog vremena i kompleksnosti, ispostavilo se da je najveći udio uzeo razvoj REST API-ja, odnosno backenda ovog sistema. U stvarnosti, nakon istraživanja raznih tekstova, od 60% do 70% razvoja BaaS sistema otpada na razvoj backenda. Dakle, i to je jedan od razloga zašto postoje mnogi,

unaprijed definisani sistemi koje programeri mogu da iskoriste te uštede resurse i vrijeme. U svakom slučaju postoje izazovi prilikom razvoja ovakvih sistema, a neki od njih su različitost klijenata za koji se kreiraju ovakvi sistemi, zahtjevi samog klijenta, konačni ciljevi, skalabilnost ili proširivost, visoke performanse i sigurnost. Na kraju, donositelji odluka u ovakvim projektima, naručioci, inžinjeri, te sama radna snaga koja radi na razvoju, moraju da imaju temeljna znanja o BaaS sistemima, njihovim arhitekturama, mogućnostima, te najvažnije - ciljevima.

LITERATURA

- [1] *A Backend-as-a-Service (BaaS) Overview*, <https://telusdigital-marketplace-production.s3.amazonaws.com/iot/user-content/product/818d-o.pdf> (14.06.2019.)
- [2] *Appcelerator platform*, <https://www.appcelerator.com/> (17.06.2019.)
- [3] *Appery.io platform*, <https://appery.io/> (17.06.2019.)
- [4] *Backend as a Service (BaaS)*, <https://www.techopedia.com/definition/29428/backend-as-a-service-baas> (15.06.2019.)
- [5] Chandrasekaran, K.: *Essentials of Cloud Computing*, CRC Press, USA, 2015
- [6] Copperwaite, M., Leifer, C.: *Learning Flask Framework*, Packt Publishing, Birmingham, 2015
- [7] Flanding, J., Grabman, G., Cox, S.: *Leading Change in the Digital Era*, Emerald Publishing, United Kingdom, 2019
- [8] *Flask*, <http://flask.pocoo.org/> (23.06.2019)
- [9] Grinberg, M.: *Flask Web Development: Developing Web Applications with Python*, O'Reilly Media, Inc., USA, 2014
- [10] Hellegouarch, S.: *CherryPy Essentials: Rapid Python Web Application Development*, Packt Publishing, Birmingham, 2007
- [11] *Heroku*, <https://www.heroku.com/> (27.06.2019.)
- [12] *JavaScript*, <https://en.wikipedia.org/wiki/JavaScript> (25.06.2019.)
- [13] *JavaScript*, <https://www.javascript.com/> (26.06.2019.)
- [14] *JetBrains*, <https://www.jetbrains.com/pycharm/features/> (22.06.2019.)
- [15] *JSON*, <https://www.json.org/> (26.06.2019.)
- [16] *Kinvey platform*, <https://www.progress.com/kinvey> (18.06.2019.)
- [17] Lane, K.: *Overview of the Backend as a Service (BaaS) Space*, API Evangelist, 2013
- [18] Nguyen, P.: *Mobile Backend as a Service: The Pros and Cons of Parse*, Lahti

University of Applied Sciences, Lahti, 2016

[19] *Parse platform*, <http://parseplatform.org/> (18.06.2019.)

[20] *Python*, <https://www.python.org/> (20.06.2019.)

[21] Rasthofer, S., Arzt, S., Hahn, R., Kolhagen, M., Bodden, E.: *(In)Security of Backend-as-a-Service*, Center for Advanced Security Research Darmstadt, Darmstadt, 2016

[22] *SQLite*, <https://www.sqlite.org/index.html> (23.06.2019.)

[23] Underdahl, B.: *Enterprise Mobility for Dummies*, John Wiley & Sons, Inc., Hoboken, 2016