Razvoj aplikacije uz pomoć arhitekture bez servera - primer upotrebe Firebase i iOS platforme

Aleksandar Dinić

***Sadržaj* — Arhitektura bez servera nudi rešenja za brži razvoj aplikacije kao i smanjenju troškova razvoja i održavanja u odnosu na klijent-server arhitekturu. Iako naziv može pogrešno da sugeriše, u arhitekturi bez servera server i dalje postoji ali o njemu brine klaud provajder. Na taj način možemo se fokusirati na razvoj i održavanje same aplikacije a ne na infrastrukturi koja je potrebna za rad iste.**

U ovom radu predstavili smo prednosti i mane ovog razvoja softvera. Razvili smo aplikaciju koja pruža informacije o stripovima koristeći upravo arhitekturu bez servera. Za klaud provajdera koristili smo Firebase i predstavili njegove proizvode koji su nam potrebni za rad ove aplikacije.

Klijentski deo aplikacije implementirali smo za iOS platformu. Za programski jezik koristili smo Swift, kao jezik novijeg datuma. Opisali smo integraciju između Firebase-a i iOS-a, kao i osnovne funkcionalnosti klijentske aplikacije.

***Ključne reči* — Arhitektura bez servera, Firebase, iOS, klijent-server arhitektura, strip, Swift.**

I. UVOD

RAZVOJ aplikacije često zahteva razvoj klijenta i servera. Na ovaj način delimo svoje zadatke na server koji obezbeđuje neke usluge i klijente koje te usluge potražuju. Trenutno je izuzetno veliki broj tehnologija na tržištu u kojima možemo razvijati obe strane našeg sistema. Ako na to dodamo i veoma brzi razvoj novih tehnologija, jasno je da moramo imati mnoštvo veština kako bi razvili celokupni sistem.

Aleksandar Dinić, Beograd, Srbija (e-mail: a.dinic@icloud.com).

Velike firme zbog toga imaju i velike timove. U timu svako radi na konkretnom delu sistema za koji je specijalizovan. Ali kako sve to organizovati i implementirati u malom timu ili kao pojedinac?

Kako bi korisnicima pružili što bolje korisničko iskustvo i što lepši korisnički interfejs, moramo dosta vremena uložiti na izradi klijenta. Svaki klijent ima određena pravila i funkcionalnosti koje korisnici očekuju da se nalaze u svakoj aplikaciji. Zbog toga moramo našu aplikaciju prilagoditi svakom klijentu. Ovo zahteva dobro poznavanje samog klijenta, navike korisnika kao i njihovo ponašanje. Takođe moramo biti u toku i sa najnovijim funkcionalnostima koje dolaze, pa se zbog toga rad na klijentu nikada ne završava.

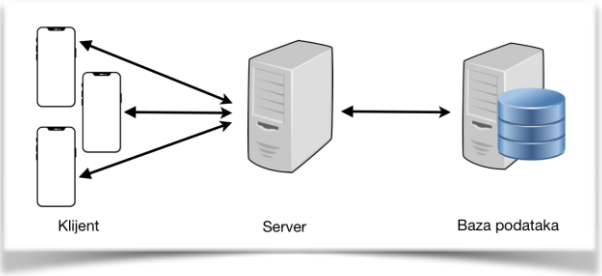
Sa druge strane ne smemo zapostaviti ni server, jer za ispravno funkcionisanje kompletnog sistema server igra značajnu ulogu. Koliko god klijent bio moderan i inovativan ukoliko server ne postiže da odgovori na sve njegove zahteve, sam klijent može biti neupotrebljiv. Pored implementacije naših funkcionalnosti na serveru imamo i dodatnog posla oko infrastrukture koja je potrebna za njegov rad. Moramo voditi računa da server radi i kada nema zahteva prema njemu, odgovorni smo za njegovu sigurnost, odgovorni smo za skaliranje usled povećanja saobraćaja itd... Svesni su ovih problema i velike kompanije (Google, Amazon, Apple…). Upravo oni i nude rešenja u arhitekturi bez servera, kako bi se manji timovi ili pojedinci fokusirali na razvoj i održavanje same aplikacije a ne na infrastrukturu koja je potrebna za rad iste.

U ovom radu pokušaćemo da kroz jednostavni primer aplikacije za pregled informacija o stripovima predstavimo sve prednosti i mane ovog načina razvijanja softvera. Koristićemo Firebase za arhitekturu bez servera, dok ćemo klijentski deo aplikacije implementirati za iOS platformu.

II. Arhitektura bez servera?

Svaka aplikacija koja skladišti podatke, odrađuje podatke i prikazuje te podatke na nekom drugom mestu možemo nazvati klijent-server aplikacijom. Server je odgovoran za skladištenje i obradu tih podataka, dok je uloga klijenta da kreira nove i prikazuje već postojeće podatke. Ideja klijent-server arhitekture je da omogući većem broju korisnika pristup istim podacima [1].

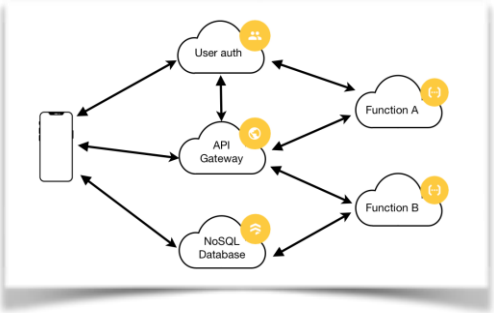
Klijent-server arhitekturu možemo vizuelno predstaviti kao na slici 1. U ovoj arhitekturi klijent (na slici predstavljen kao mobilna aplikacija) je odgovoran za korisnički interfejs i obradu akcija korisnika, ali većinu biznis logike potrebno je da delegira serveru. Zahtevi koji dolaze sa strane klijenta server obrađuje, komunicira sa bazom (ukoliko je to potreba konkretnog zahteva) i odgovor vraća klijentu na dalju upotrebu.



Sl. 1. Klijent-Server arhitektura

Da bi kreirali ovakvu aplikaciju moramo imati širok spektar znanja i veština. Klijentski deo je ujedno identitet naše aplikacije, jer je to i jedini deo aplikacije kojem korisnici imaju pristup. Pored klijenta moramo razviti, konfigurisati i upravljati serverom, takođe moramo konfigurisati i upravljati bazom podataka. Čak i nakon završetka implementacije servera i baze podataka moramo i dalje konfigurisati i upravljati njihovim sistemima. Ne smemo zaboraviti ni rad na sigurnosti, skaliranju usled povećanja/smanjenja saobraćaja kao i dostupnosti u svakom trenutku. Implementacija ovih sistema je svakako moguća i veoma izazovna, ali za pojedinca ili mali tim može biti previše kompleksna, čak do te mere da ostavlja malo prostora za ispravljanje postojećih problema, dodavanja novih funkcionalnosti ili razvoj novih ideja [2].

Kao rešenje ovih problema nastala je ideja o arhitekturi bez servera (serverless arhitecture). Kao i kod mnogih trendova u softveru ne postoji stroga definicija šta se sve podrazumeva pod arhitekturom bez servera. U početku ovaj termin je korišćen da bi opisao aplikacije čiji se server nalazio na nekom od klaud provajdera. Ove aplikacije su takođe mogle da koriste i ostale pogodnosti koje provajder nudi, poput baze podataka, autentifikacije itd... Drugi naziv za ovu vrstu implementacije je (Mobile) Backend as a Service, ili BaaS. Arhitektura bez servera se takođe odnosi i na rešenja gde se serverska logika nalazi na klaud provajderu, ali za razliku od tradicionalnih rešenja server implementiramo uz pomoć funkcija koje se aktiviraju uz pomoć nekog događaja. Ovaj pristup se takođe naziva Functions as a Service, ili FaaS. BaaS i FaaS se često koriste zajedno tako da klaud provajderi nude pogodnosti korišćenja i jednog i drugog pristupa [3]. Na slici 2. možemo videti jednu od implementacija arhitekture bez servera.



Sl. 2. Arhitektura bez servera

Na ovom primeru korisnički interfejs se i dalje nalazi na klijentskom delu aplikacije dok se autentifikacija i menadžment korisnika nalazi na jednom od BaaS proizvodu (User auth). Ovaj servis se može pozvati direktno sa klijenta prilikom registracije ili logovanja ali se takođe može pozvati i iz drugog dela sistema kako bi se dobili podaci o korisnicima.

U klijent-server arhitekturi celokupnu logiku oko autentifikacije i menadžmenta korisnika morali smo sami da implementiramo, dok u ovom pristupu logika se nalazi na BaaS. Ovaj pristup se koristi i za ostale proizvode. Trebamo dodati da prilikom određenih akcija, poput kreiranje novog korisnika, upisivanje/brisanje iz baze itd, možemo aktivirati željene funkcije. Na ovaj način možemo određenu logiku implementirati na klijentu koju smo ranije morali implementirati na serveru.

Ovako novokreirana arhitektura bez servera izgleda znatno komplikovanije od klijent-server arhitekture. Ipak na ovaj način smanjili smo potrebu za upravljane infrastrukturom već svoje vreme možemo usmeriti na razvoj funkcionalnosti naše aplikacije [2].

A. Prednosti arhitekture bez servera

Arhitekturom bez servera ne moramo trošiti vreme na monitoring rada našeg servera, ovaj posao ostavljamo klaud provajderu dok se mi fokusiramo isključivo na domensku logiku naše aplikacije. Na taj način smanjujemo troškove izrade aplikacije. Takođe ne moramo implementirati funkcionalnosti koje su već uveliko rešene, poput autentifikacije korisnika, puš notifikacija itd.

Usled povećanja broja korisnika naša aplikacija zahteva i više resursa za rad kako bi u doglednom vremenu odgovorila na zahteve korisnika. U sopstvenoj implementaciji sami smo odgovorni za kupovinu/zakup dodatnog hardvera kako bi to postigli. Arhitekturom bez servera klaud provajder se brine o tome.

Takođe naplaćuje se samo onoliko koliko je aplikaciji stvarno potrebno za rad pa samim tim ukoliko dođe do manjeg broja korisnika i cena će se automatski smanjiti.

Važno je spomenuti i sigurnost usled potencijalnih napada o kojem se takođe brine klaud provajder. Tu dolazi i sigurnost samih podataka sa kojima naša aplikacija raspolaže [4], [5].

B. Ograničenja arhitekture bez servera

Jedna od loših strana arhitekture bez servera je svakako kompleksnost same arhitekture. Videli smo na primeru da je dosta komplikovanija od klijent-server pristupa pa zahteva i određeno vreme upoznavanja/učenja.

U potpunosti smo zavisni od klaud provajdera i u koliko iz bilo kog razloga provajder prestane sa radom moramo preći na neko drugo rešenje. Svi naši podaci i podaci naših korisnika su takođe na raspolaganju klaud provajderu. Ukoliko je potreba naše aplikacije da se podaci ne daju na uvid trećoj strani arhitektura bez servera nije dobar izbor.

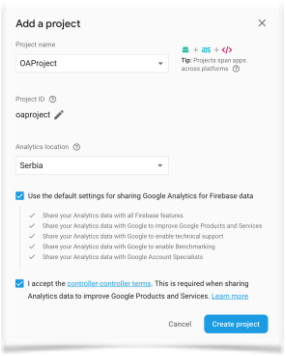
Lokalno testiranje je još jedan nedostatak ovog pristupa. Testove je svakako moguće pisati za svaki proizvod ponaosob, ali testirati celokupne funkcionalnosti u lokalu može biti znatno otežano [4], [5].

III. IMPLEMENTACIJA SERVERA

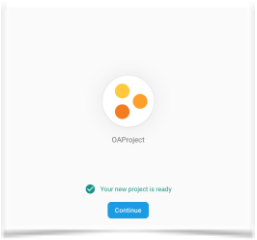
Firebase poseduje izuzetno dobro napisanu zvaničnu dokumentaciju [6] kao i mnoštvo zvaničnih primera [7] koji u znatnoj meri pomažu u implementaciji. U ovom poglavlju opisaćemo kako se kreira Firebase nalog, kao i tri proizvoda (Cloud Functions, Cloud Firestore i Firebase Authentication) koja su nam potrebna za rad naše aplikacije.

A. Kreiranje Firebase naloga

Da bi krenuli sa podešavanjem firebase-a moramo se pre svega ulogovati na firebase.google.com sa bilo kojim Google nalogom. Prva stvar je kreiranje novog projekta koji mora da poseduje jedinstveno ime na firebase-u. Nakon uspešnog kreiranog projekta (Slika 4) odlazimo na konzolu (Slika 5) i možemo krenuti sa implementacijom naših servisa.



Sl. 3. Kreiranje novog projekta



Sl. 4. Uspešno kreiran projekat

Kao što možemo videti već na početku imamo izbor da dodamo firebase na našem klijentu gde su podržani iOS, Android, Web i Unity. U ovom trenutku nećemo razmišljati o klijentu, sasvim je u redu da izaberemo bilo koji od ponuđenih, jer našu implementaciju krećemo od backend dela.

Sl. 5. Firebase konzola

B. Cloud Functions

Za implementaciju backend dela potreban nam je Cloud Functions [8]. Cloud Functions je deo Google cloud platforme. On nam omogućava da pišemo i implementiramo kod koji se automatski izvršava kao odgovor na događaje izazvane Firebase proizvoda ili web zahtevima. Ovako implementiran kod se nalazi na Google serverima i na taj način celokupan kod nam je centralizovan i Google garantuje za njegovu sigurnost. Firebase se takođe brine o skaliranju koda da ne dolazi do zagušenja prilikom povećanja saobraćaja ili prevelikih troškova kada saobraćaj ne postoji.

Sl. 6. Cloud Functions kao centralni deo Google cloud platforme

Za rad Cloud Functions potrebno nam je Node.js [9] okruženje i TypeScript [10]. Tačnije možemo izabrati JavaScript ili TypeScript, ali u ovom radu mi koristimo TypeScript. Pre nego što napišemo prvu liniju koda, moramo instalirati Node.js, Firebase CLI [11] (command line interface) i code editor.

Postoji nekoliko načina kako sve ovo možemo instalirati. Iz ličnog afiniteta u ovom radu koristićemo Homebrew [12]. Pre svega instaliraćemo node uz pomoć komande:

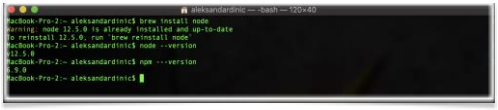
```
$ brew install node
```

U svakom trenutku možemo proveriti koju verziju noda imamo instaliranu:

```
$ node —version
```

Firebase za rad zahteva minimalnu verziju 6.0.0. Node instalacija bi trebalo da sadrži i npm (node package manager), gde je nama potrebna verzija 5.0.0 ili novija. Instaliranu verziju možemo proveriti sa:

```
$ npm --version
```



Sl. 7. Instalacija Node.js

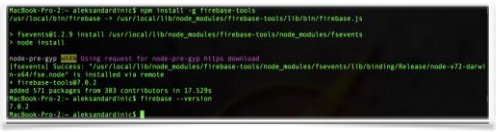
Nakon uspešne instalacije Noda, možemo instalirati Firebase CLI.

```
$ npm install -g firebase-tools
```

Kada se instalacija završi, komandom:

```
$ firebase --verison
```

možemo proveriti verziju. Poželjno je da uvek imamo najnoviju verziju.



Sl. 8. Instalacija Firebase CLI

Sada imamo instaliran celokupni softver koji nam je potreban za rad sa Cloud Functions. Poslednja stvar koja nam je potrebna je code editor za TypeScript i u ovom radu ćemo koristiti Visual Strudio Code (VS Code) [13].

Instalaciju TypeScript-a radimo uz pomoć komande:

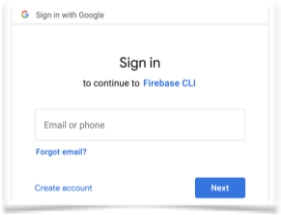
```
$ npm install -g typescript
```

Kreiraćemo folder OAProject gde će se nalaziti naš kod i sada je potrebno da se ulogujemo na Firebase uz pomoć komande:

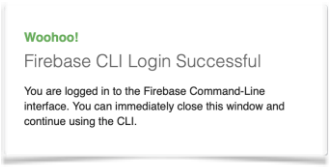
```
$ firebase login
```

Sl. 9. Firebase login

Otvoriće nam se forma u browser za logovanje u kojoj možemo da se ulogujemo sa bilo kojim Google nalogom, ali potrebno je da se ulogujemo sa nalogom sa kojim smo kreirali Firebase (Slika 10). Nakon uspešnog logovanja dobijamo potvrdu (Slika 11).



Sl. 10. Forma za logovanje



Sl. 11. Potvrda uspešnog logovanja

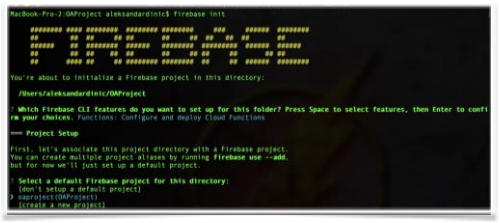
Dalji rad nastavljamo u konzoli gde komandom:

```
$ firebase init
```

kreiramo Firebase projekat i prvi izbor je izbor proizvoda koje želimo da koristimo. Možemo izabrati više proizvoda od jednom, za sada izabraćemo samo Functions, kasnije po potrebi možemo dodati ostale proizvode.

Sl. 12. Kreiranje Firebase projekta

Drugi izbor je izbor default projekta, gde ćemo izabrati već kreirani OAProject (Slika 13).



Sl. 13. Izbor default projekta

Zatim izbor programskog jezika u kojem želimo da pišemo naše funkcije, gde kao što smo rekli izabraćemo TypeScript (Slika 14).

Sl. 14. Izbor programskog jezika

Naredni izbor je da li želimo da koristimo TSLint za hvatanje bug-ova, svakako može biti od pomoći (Slika 15).

Sl. 15. TSLint

Poslednja opcija je da li želimo da instaliramo dependency sa npm, naravno da želimo, jer ove module koristimo za pisanje svojih funkcija (Slika 16).

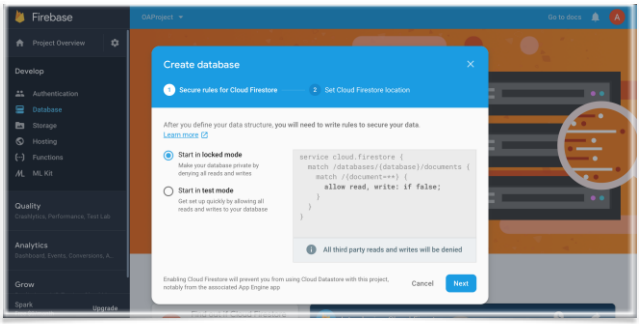
Sl. 16. Instalacija dependency sa npm

Nakon toga završili smo instalaciju Firebase projekta (Slika 17).

Sl. 17. Uspešno instaliran Firebase projekat

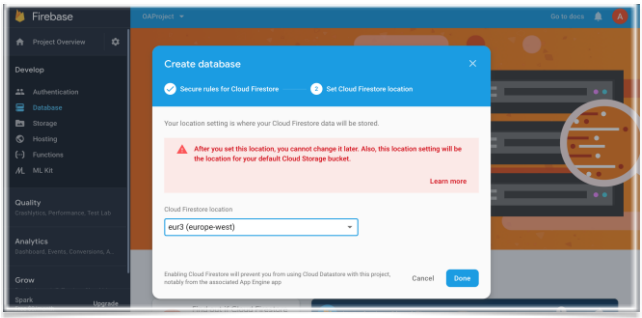
C. Cloud Firestore

Cloud Firestore [14] je proizvod za rad sa bazom podataka. Pre nego što krenemo sa korišćenjem ovog proizvoda moramo prvo kreirati bazu podataka. Dobijamo kratko objašnjenje kako sigurnosna pravila funkcionišu i na nama je izbor da li krećemo u locked ili test mode (Slika 18).



Sl. 18. Kreiranje baze podataka

Sve jedno je u kom režimo sada kreiramo našu bazu podataka, jer svakako je definisanje ovih pravila jedno od prvih stvari koje ćemo implementirati. Drugi korak prilikom kreiranje baze je izbor lokacije gde će se naši podaci nalaziti. Izabraćemo eur3 (europe-west), iz Multi-region (Slika 19). Prednost multi-region u odnosu na region je da su podaci duplirani na više regiona, a unutar regione su duplirani u više zona. Na taj način znatno povećavamo dostupnost i bezbednost podataka. Ukoliko dođe do gubitka čak i celog regiona, naši podaci su sigurni jer se nalaze na više mesta.

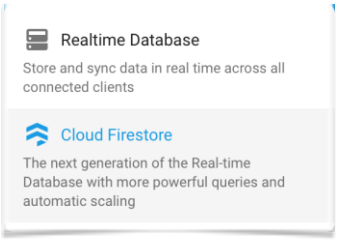


Sl. 19. Izbor lokacije

Nakon što smo uspešno kreirali bazu podataka, možemo izabrati između

Realtime Database i Cloud Firestore (Slike 20). Naš izbor je svakako Cloud Firestore iz razloga što u zvaničnoj dokumentaciji stoji da je to novija generacija Firebase baze podataka koja poseduje novije i bolje funkcionalnosti u odnosu na Realtime Database.

Cloud Firestore kao i svi ostali proizvodi dolaze sa besplatnom kvotom na dnevnom nivou (Slika 21), nakon te kvote plaćamo onoliko koliko smo koristili. Ovo je jedna od stavki gde možemo stvoriti bespotrebne troškove ukoliko imamo lošu implementaciju. Ovo je pravo mesto da razmislimo o organizaciji i keširanju naših podataka.

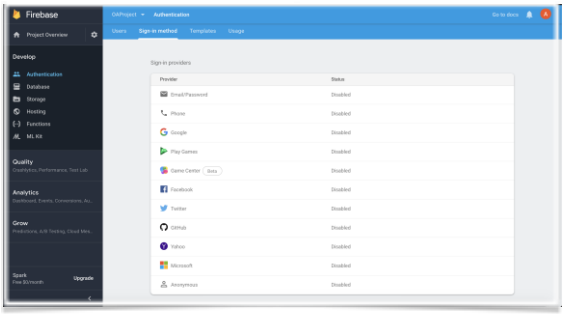


Sl. 20. Izbor baze

Sl. 21. Besplatna kvota na dnevnom nivou

C. *Firebase Authentication*

Naša aplikacija ima potrebu da zna identitet svojih korisnika. Sa ovim informacijama možemo personalizovati korisničko iskustvo, ponuditi beneficije lojalnijim korisnicima, vršiti A/B testiranje sa targetiranim korisnicima itd...

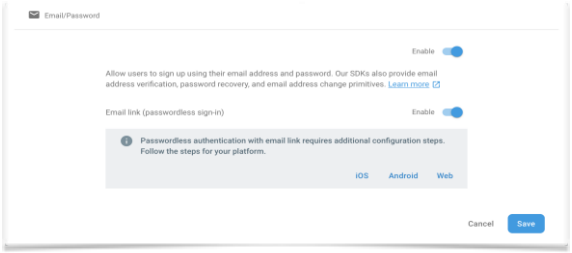


Sl. 22. Firebase Authentication

Jedna od funkcionalnosti koje želimo da ponudimo unutar naše aplikacije je da korisnicima damo mogućnost katalogizacije svojih stripove. Ovu funkcionalnost bez problema možemo implementirati na svakom od klijenta

ali problem nastaje ukoliko korisnik pređe sa jednog uređaja na drugi. Ukoliko korisnik nije registrovan, sve ono što je uradio na starom uređaju neće postojati na novom.

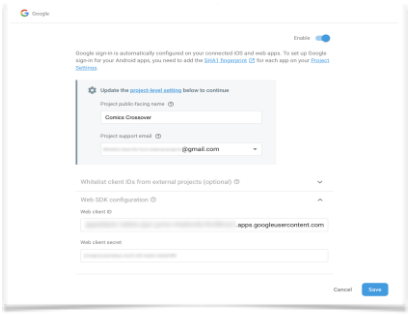
Uz pomoć Firebase Authentication [15] korisnicima možemo ponuditi mogućnost registracije i logovanja na naš sistem. Potrebno je samo da izaberemo sa kojim provajderima korisnici mogu da se loguju. Email i password je svakako ono što želimo i za ovu opciju nemamo dodatnih podešavanja, dovoljno je samo da je aktiviramo.



Sl. 23. Email Password provajder

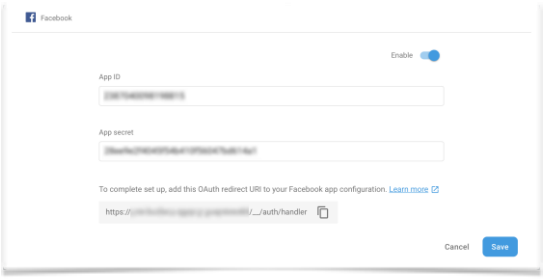
Spisak ostalih provajdera je impozantan, tako da korisnicima možemo omogućiti da se veoma jednostavno registruju na naš sistem. Teoretski možemo korisnicima da ponudimo sve ove opcije za registraciju, ali to nije najbolje rešenje jer korisnik posle nekog vremena može lako zaboraviti sa kojim provajderom je izvršio registraciju. U ovom radu implementiraćemo Google, Facebook i Anonymous.

Za implementaciju Google autentifikacije podešavanja su veoma jednostavna. S obzirom da je Firebase u vlasništvu Google, svi podaci koje treba da popunimo su automatski popunjeni za nas, potrebno je samo da aktiviramo ovu opciju.



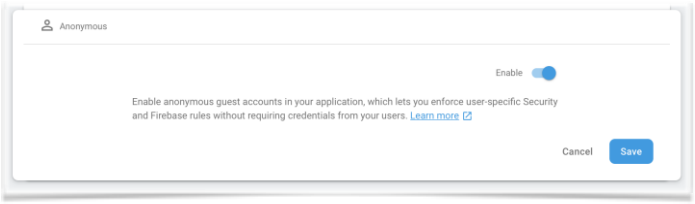
Sl. 24. Google provajder

Kako bi omogućili registraciju korisnika uz pomoć Facebook-a, potrebni su nam App ID i App secret. Da bi dobili ove informacije potrebno je da odemo na developers.facebook.com, ulogujemo se sa bilo kojim Facebook profilom i kreiramo aplikaciju. Za kreiranje aplikacije potrebno je da unesemo samo njeno ime i kontakt email, sve ostale informacije možemo uneti i kasnije. Nakon uspešno kreirane aplikacije, Facebook dodeljuje App ID i App secret tako da nam ostaje samo da ove podatke upišemo u Firebase konzolu.



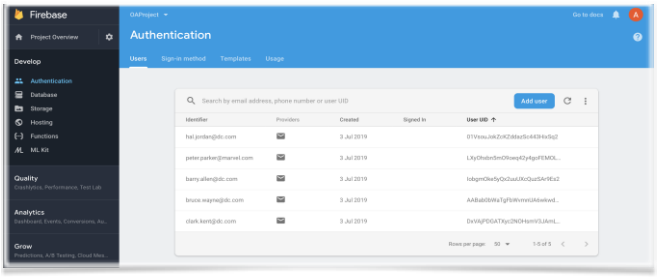
Sl. 25. Facebook provajder

Poslednji provajder kojeg dodajemo je Anonymous. Moramo priznati da je registracija poprilično zamorna za sve korisnike. Kada prvi put otvorimo novu aplikaciju nismo ni sigurni šta ta aplikacija uopšte radi. Zbog toga nema potrebe primoravati korisnika da se registruje ali trebamo mu prikazati koje su to pogodnosti ukoliko se registruje. Sve dok se ne registruje trebamo tom korisniku omogućiti da koristi aplikaciju potpuno neometano. Zbog toga aktiviraćemo anonimnu registraciju gde registrujemo korisnika prilikom prve njegove posete, pratimo njegove aktivnosti, bez ikakve informacije o samom korisniku. Kada korisnik odluči da se registruje, njegov anonimni profil sprovedemo u registrovanog korisnika sa svim aktivnostima do tada. Ukoliko korisnik ipak ne odluči da se registruje a pritom promeni uređaj sa koga koristi našu aplikaciju, sve njegove aktivnosti će nažalost biti izgubljene.



Sl. 26. Anonymous provajder

Sa ovim smo završili sa aktivacijom željenih provajdera, završili smo kompletno podešavanje autentifikacije na backend strani. U firebase konzoli možemo dodati nove korisnike kao što vidimo na slici 27.

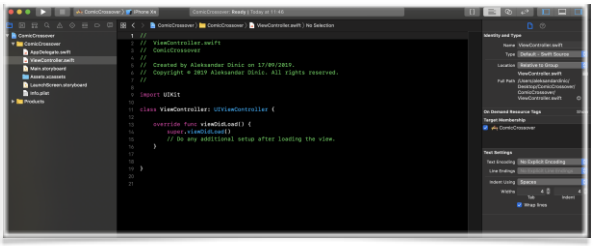


Sl. 27. Registrovani korisnici

IV. Implementacija klijenta

U prvoj verziji naše aplikacije razvićemo klijentski deo aplikacije za iOS platformu. iOS je operativni sistem koji je razvila kompanija Apple. Ovaj operativni sistem je originalno razvijen za pametne telefone iPhone ali je kasnije dodat i na iPod touch kao i na iPad uređaje. Kako bi razvijali aplikacije za iOS moramo pre svega imati Apple računar kao i Xcode razvojno okruženje.

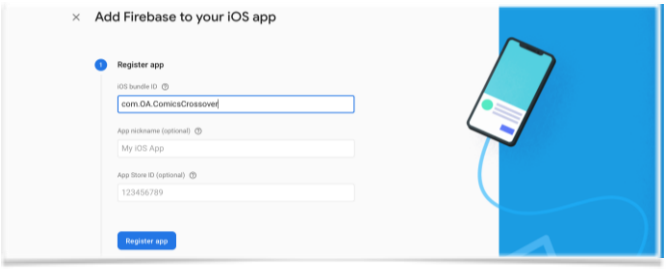
Xcode takođe razvija kompanija Apple, na slici 28 možemo videti kako izgleda ovo razvojno okruženje. Za razvoj iOS imamo u ponudi dva programska jezika Objective-C i Swift. Moguće je u istom projektu kombinovati ova dva jezika koja bez ikakvih problema funkcionišu zajedno. U ovom radu kompletnu aplikaciju razvijaćemo u jeziku Swift koji je jezik novijeg datuma sa mnogo više funkcionalnosti u odnosu na Objecitve-C.



Sl. 28. Xcode razvojno okruženje

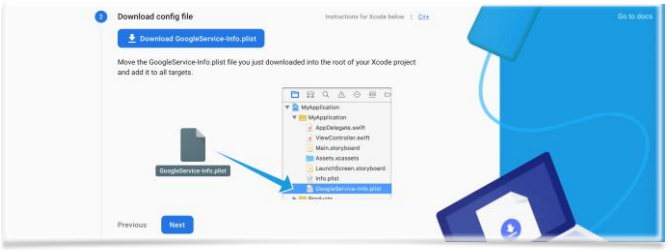
A. Integracija Firebase-a i iOS aplikacije

Nakon kreiranog novog projekta potrebno je da dodamo firebase u našoj aplikaciji [16]. Ovo postižemo tako što u firebase konzoli najpre registrujemo novu aplikaciju. Za to nam je potreban bundle identifier (bundle ID) koji jedinstveno određuje našu aplikaciju.



Sl. 29. Prvi korak registracija aplikacije

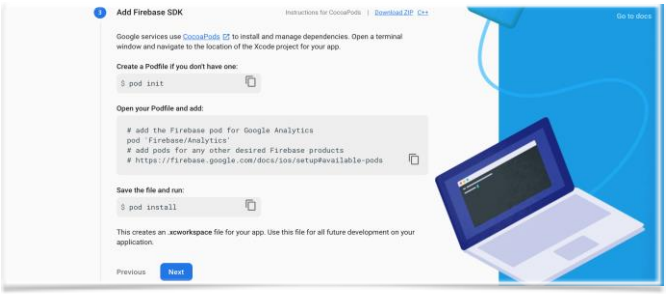
Kao drugi korak potrebno je dodati konfiguracioni fajl sa nazivom GoogleService-Info.plist koji firebase generiše za nas. Ovaj fajl dodajemo u naš projekat i u njemu se nalaze informacije poput id klijenta, api ključa itd. koji su potrebni prilikom komunikacije firebase i naše aplikacije.



Sl. 30. Drugi korak konfiguracioni fajl

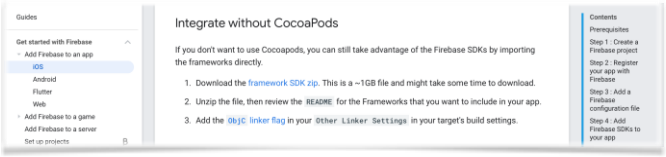
Treći korak je dodavanje firebase SDK u aplikaciji. Ovo možemo uraditi na dva načina. Preporuka je da se SDK doda preko CocoaPods kao jedan od najpopularniji dependency menadžer za Swift i Objective-C. Iako CocoaPods izuzetno dobro radi posao za koji je namenjen mi u ovom radu ga nećemo koristiti. Razlog tome je kompleksnost koju CocoaPods donosi. CocoaPods kreira Workspace koji u sebi može sadržati više projekta. Gde je jedan projekat naš projekat koji smo kreirali dok je drugi projekat Pods koji vodi računa o dependency između biblioteka. Pored toga CocoaPods promeni i određene konfiguracione fail-ove za svoj rad koje nije lako ispratiti. Sve ovo možemo i

mi sami kreirati i podesiti. Potrebno je da uložimo neko vreme u podešavanju projekta ali u slučaju bilo kog problema možemo detektovati i ukloniti problem.



Sl. 31. Treći korak firebase SDK

Na slici 32 možemo videti kako da dodamo firebase SKD bez CocoaPods-a. Potrebno je da skinemo SDK zip fajl i dodamo framework koje želimo, s tim da je Analytics framework obavezan. Na ovome vidimo lošu stranu korišćenja već postojećih rešenja. Da smo se odlučili za CocoaPods možda bi ova zavisnost prošla i nezapažena sa naše strane ali na to smo pristali izborom arhitekture bez servera.

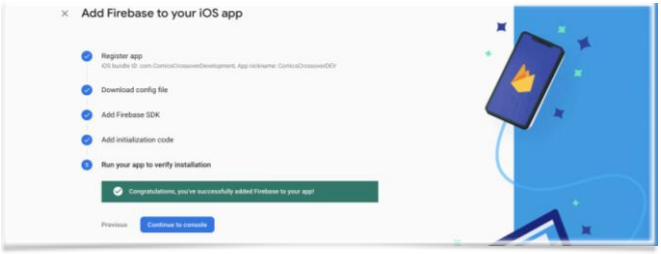


Sl. 32. Firebase SDK bez CocoaPods-a

Naredni korak je inicijalizacija firebase unutar naše aplikacije. Dovoljno je da dodamo deo koda kao na primeru u AppDelegate klasi.

Sl. 33. Inicijalizacija Firebase-a

Na kraju potrebno je da pokrenemo aplikaciju. Nakon uspešnog povezivanja dobijamo potvrdu kao što možemo videti na slici 34.

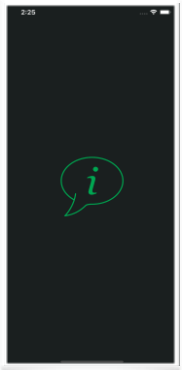


Sl. 34. Potvrda uspešne integracije firebase-a i iOS aplikacije

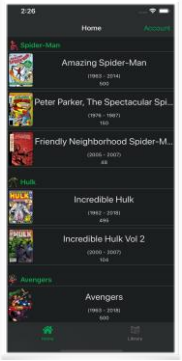
B. Funkcionalnost aplikacije

Početni ekran prilikom pokretanja aplikacije je takozvani Launch Screen. Ovaj ekran se brzo zamenjuje prvim ekranom što daje utisak da je aplikacija brza. Jedina namena ovog ekrana je da ostavi utisak da se aplikacija brzo pokreće i da je odmah spremna za upotrebu. Ukoliko se aplikacija pokrene dovoljno brzo ovaj ekran može proći i nezapaženo. Zbog toga ovaj ekran ne sadrži previše elemenata već kao što možemo videti na slici 35 sadrži samo logo naše aplikacije.

Na prvom ekranu aplikacije (slika 36) prikazani su serijali koji su grupisani po karakterima. Karakteri kao i serijali unutar svakog karaktera su sortirani po popularnosti. Ukoliko popularnost ne postoji ili je iste vrednosti sortiraju se po svom nazivu. Kao što možemo videti, na ovom ekranu dostupni su nam samo imena karaktera kao i njihove slike. Dok kod serijala pored naziva i slika imamo i broj stripova u samom serijalu kao i godinu početka i završetka serijala.



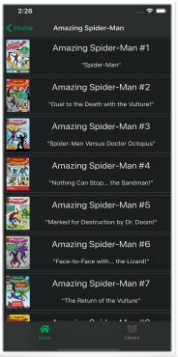
Sl. 35. Launch Screen



Sl. 36. Karakteri sa serijalima

Klikom na serijal odlazimo na drugi ekran koji prikazuje listu stripova unutar serijala (slika 37). Na ovom ekranu možemo videti listu svih stripova u ovom serijalu gde su od podataka dostupni ime stripa, slika kao i datum izdanja. Za ostale informacije o konkretnom stripu potrebno je da kliknemo na željeni strip i samim tim odlazimo na treći ekran (slika 38).

Na ovom ekranu dobijamo sve informacije o izabranom stripu. U prvom planu je naslovna strana stripa koja zauzima veliki deo ekrana. Pored naziva, datuma izlaska tu su i opis radnje kao i imena kreatora koji su stvorili ovaj strip.



Sl. 37. Stripovi u jednom serijalu

Sl. 38. Informacije konkretnog stripa

LITERATURA

[1] George Reese, "Database Programming with JDBC and Java", ISBN-13: 978-1565922709
[2] Michael Roberts and John Chapin, "What Is Serverless?", ISBN: 9781491984178
[3] Serverless Architectures. Available: martinfowler.com/articles/serverless.html
[4] Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Suter, P. (2017). "Serverless Computing: Current Trends and Open Problems. Research Advances in Cloud Computing", 1–20. doi:10.1007/978-981-10-5026-8_1
[5] Baldini, I., Castro, P., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Suter, P. (2016). "Cloud-native, event-based programming for mobile applications. Proceedings of the International Workshop on Mobile Software Engineering and Systems" - MOBILESoft '16. doi:10.1145/2897073.2897713
[6] Firebase dokumentacija. Available: firebase.google.com/docs
[7] Firebase github. Available: github.com/firebase
[8] Firebase Cloud Functions. Available: firebase.google.com/docs/functions
[9] Node.js. Available: nodejs.org/about
[10] TypeScript. Available: www.typescriptlang.org
[11] Firebase Command Line Interface (CLI). Available: firebase.google.com/docs/cli
[12] Homebrew. Available: brew.sh
[13] Visual Strudio Code (VS Code). Available: code.visualstudio.com/docs
[14] Firestore Firebase. Available: firebase.google.com/docs/firestore
[15] Firebase Authentiction. Available: firebase.google.com/docs/auth
[16] Integracija Firebase i iOS projekta. Available: firebase.google.com/docs/ios/setup

ABSTRACT

Serverless architecture offers solution for faster application development as well as reducing development and maintenance costs relative to client-server architecture. Although the name might be misinterpreted, in serverless architecture, the server still exists but it is taken care of by the cloud provider. In that manner, we can focus on the development and maintenance of the application itself, rather than on the infrastructure required to run the application.

In this paper we have presented advantages and disadvantages of serverless software development. We have developed an application that provides comic book information using serverless architecture. For the cloud provider, we used Firebase and introduced its products that are needed to run the application.

We have implemented the client part of the application for the iOS platform. We used Swift, as a programming language, described the integration between Firebase and iOS, as well as the basic functionality of the client application.

Application development using serverless architecture - An example of using Firebase and the iOS platform

Aleksandar Dinić