Tehnologije registracije i prijave s implementacijom u Kotlinu

Ivanović, Toni

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Toni Ivanović

TEHNOLOGIJE REGISTRACIJE I PRIJAVE S IMPLEMENTACIJOM U KOTLINU

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Toni Ivanović

Matični broj: 0016152212

Studij: Informacijski i poslovni sustavi

TEHNOLOGIJE REGISTRACIJE I PRIJAVE S IMPLEMENTACIJOM U KOTLINU

ZAVRŠNI RAD

Mentor:

Izv. prof. dr. sc. Zlatko Stapić

Varaždin, srpanj 2024.

Toni Ivanović

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Osnovna tema ovog rada je OAuth 2.0 protokol, ključni protokol kada je riječ o autorizaciji korisnika pri zahtjevu za pristup zaštićenim resursima aplikacije. S obzirom na to da je autorizacija samo dio procesa prijave, u radu su detaljno obrađeni cjelokupni procesi registracije i prijave u aplikaciju, naglašavajući njihovu međuovisnost. Za funkcioniranje OAuth 2.0 protokola u praksi neophodan je i autentifikacijski protokol OpenID Connect. Kako bi se dao bolji uvid u već spomenute procese ovi protokoli uspoređeni su sa tradicionalnijim pristupom autentifikaciji i autorizaciji korisnika, odnosno sa SAML protokolom.

Nadalje se u radu opisuju tri tehnologije koje podržavaju OAuth 2.0 i OpenID Connect protokole odnosno Firebase Authentication, Single Sign-On tehnologije te Auth0. Te tehnologije implementirane su u kroz tri zasebne Android aplikacije pisane u Kotlin programskom jeziku što se obrađuje kroz poglavlje o implementaciji te tako rad pruža cjelokupan opis i primjenu procesa registracije i prijave u Android aplikaciju.

Ključne riječi: registracija, prijava, autorizacija, autentifikacija, OAuth 2.0, Kotlin

Sadržaj

Sadržaj iii
1. Uvod 1
2. Metode i tehnike rada 2
3. Procesi pristupa aplikaciji 3
3.1. Registracija 3
3.2. Prijava 3
3.2.1. Identifikacija 4
3.2.2. Autentifikacija 4
3.2.2.1. Autentifikacija putem biometrijskih podataka 5
3.2.2.2. Višerazinska autentifikacija 6
3.2.2.3. Autentifikacija putem trećih strana 6
3.2.3. Autorizacija 7
4. Protokoli za autentifikaciju i autorizaciju 9
4.1. Open Authorization 2.0 (OAuth 2.0) 9
4.1.1. Zašto koristiti OAuth 9
4.1.2. Uloge u OAuth-u i njihova komunikacija 10
4.1.3. Načini dobivanja odobrenja za autorizaciju 11
4.1.3.1. Način korištenjem autorizacijskog kôda 12
4.1.3.2. Implicitni način 12
4.1.3.3. Način korištenjem vjerodajnica vlasnika resursa 13
4.1.3.4. Način korištenjem vjerodajnica klijenta 13
4.1.4. Pristupni token 13
4.1.4.1. Metode slanja Bearer tokena 14
4.1.4.2. Atributi u Bearer tokenu 14
4.1.5. Osvježavajući token 15
4.1.6. Klijent 16
4.1.6.1. Registracija klijenta 16
4.1.6.2. Tip i profil klijenta 17
4.1.7. OAuth u mobilnim aplikacijama 17
4.1.7.1. PKCE protokol 17
4.1.7.2. Registracija i autentifikacija mobilnih klijenata 18

4.2. OpenID Connect (OIDC) 19
4.2.1. OIDC u suradnji s OAuth protokolom 19
4.2.1.1. ID token 19
4.2.1.2. Sigurnosni aspekti OIDC protokola u suradnji s OAuth protokolom 20
4.3. Security Assertion Markup Language (SAML) 21
4.3.1. Principi rada SAML-a 21
4.3.2. Struktura SAML poruka 22
4.3.3. SAML ili OAuth s OIDC-om 22
5. Tehnologije koje podržavaju OAuth 23
5.1. Firebase Authentication 23
5.1.1. Metode autentifikacije 23
5.1.2. Zašto koristiti Firebase Authentication 24
5.2. SSO tehnologije 24
5.3. Auth0 25
5.4. Koju tehnologiju izabrati 26
6. Implementacija tehnologija za autentifikaciju i autorizaciju u mobilnu aplikaciju 28
6.1. Implementacija Firebase Authenticationa 29
6.1.1. Kreiranje novog projekta u Firebase konzoli 30
6.1.2. Implementacija početne metode za prijavu i registraciju 31
6.1.3. Implementacija prijave koristeći Google račun putem Firebase Authentication tehnologije 35
6.2. Implementacija SSO tehnologija 44
6.2.1. Implementacija prijave putem Google računa 44
6.2.2. Implementacija prijave putem Facebook računa 49
6.2.3. Osvrt na SSO implementacije 53
6.3. Implementacija Auth0 gotovog rješenja 54
6.3.1. Kreiranje i postavljanje nove Auth0 aplikacije 55
6.3.2. Integracija Auth0 aplikacije s Android aplikacijom 56
6.3.3. Osvrt na Auth0 tehnologiju 60
6.4. Osvrt na implementacije svih tehnologija 61
7. Zaključak 62
Popis literature 64

Popis slika 67

Popis tablica 68

Popis programskih kôdova 69

1. Uvod

Danas više nego ikada koristimo aplikacije i digitalne proizvode u svakodnevnom životu. Proces registracije i prijave često je prvi korak pri korištenju tih proizvoda, omogućavajući korisnicima pristup zaštićenim dijelovima aplikacije u samo nekoliko klikova. Iako su ovi procesi postali rutinski i uvelike su se standardizirali u modernim aplikacijama oni su evoluirali paralelno s digitalizacijom društva, prilagođavajući se novim sigurnosnim zahtjevima i očekivanjima korisnika. Upravo iz toga i proizlazi motivacija za izradu ovog rada. Jer iako često ne razmišljamo o ovim procesima na njima se temelje gotovo sve moderne i uspješne aplikacije.

Iako se ovi procesi čine apstraktnima po prirodi i možda teško primjenjivima izvan digitalnog okruženja, zapravo ovi procesi se nalaze svuda oko nas. Registraciju možemo zamisliti kao rezervaciju stola u restoranu. Dovoljno je dati svoje ime i prezime uz očekivano vrijeme dolaska i restoran će za nas čuvati odabrani stol. Prijava bi u ovom slučaju bila dolazak u restoran i dobivanje pristupa stolu koji smo rezervirali koristeći iste podatke koje smo dali tokom rezervacije. Tako smo kroz nekoliko jednostavnih koraka pristupili nečemu što je do tada nama bilo teoretski nedostupno. Na sličan način ovi procesi funkcioniraju i u digitalnom svijetu.

Cilj ovog završnog rada je detaljno opisati proces registracije i prijave u mobilnim aplikacijama, ali i pogledati iza procesa u sigurnosne protokole koji podupiru njihov rad, a kojima se osigurava sigurnost korisničkih podataka. Posebna pažnja posvećena je modernim i sigurnosnim protokolima koji su danas postali standardni uključujući OAuth 2.0 i OpenID Connect protokole te Security Assertion Markup Language protokolu koji je svojevrsna preteča modernijih protokola. Također analizirati će se i moderne tehnologije za implementaciju sigurnosnih protokola u mobilnim aplikacijama, kao što su Firebase Authentication, Single Sign-On tehnologije i Auth0 kao gotovo i centralizirano rješenje za implementaciju ovih funkcionalnosti u Android mobilnu aplikaciju. Konačno, kako ne bi sve ostalo na teoriji, prikazati će se i implementacija spomenutih tehnologija u Kotlin programskom jeziku, a na kraju samog rada bit će izveden i zaključak o cjelokupnoj temi.

2. Metode i tehnike rada

Za izradu ovog završnog rada korištene su informacije iz knjiga, ali i iz online baza poput znanstvenih i stručnih članaka, dokumentacije za programere i ostalih izvora vezanih uz temu rada. Kvalitativna metoda analize korištena je za analizu literature i programskih kôdova važnih za razumijevanje teme, a metoda sinteze za donošenje i predstavljanje zaključaka o temi.

Tehnički alati korišteni za implementaciju su Android Studio kao integrirano razvojno okruženje za rad na Android mobilnim aplikacijama te Kotlin programski jezik u kojem su implementacije kôda pisane. Za dizajn korisničkog sučelja korišten je Jetpack Compose. Za verzioniranje i objavu kôda korišten je Git alat i platforma GitHub, a izvorni kôd dostupan je na javnom repozitoriju koji se nalazi na poveznici: https://github.com/tivanovic21/foi-zavrsni/.

Uz tehničke alate tehnologije korištene u implementaciji jesu Firebase Authentication, Single Sign-On tehnologije poput Google i Facebook OAuth aplikacija te Auth0 platforma.

Za stilsko oblikovanje programskog kôda korištena je Highlight Code web aplikacija dostupna na poveznici: https://highlight.hohli.com/.

3. Procesi pristupa aplikaciji

Procesi pristupa aplikaciji su razni ali uvijek spadaju u jednu od dvije kategorije: registraciju ili prijavu. Kroz ovo poglavlje teorijski ću obraditi ova dva pojma, istaknuti njihov značaj u modernim aplikacijama te kada je moguće sistematizirati potprocese od kojih su sačinjeni. Također ovisno o mogućnosti izdvojit ću raznovrsne metode za implementaciju tih procesa u samu aplikaciju, a praktična implementaciji procesa bit će opisana u zasebnom poglavlju.

3.1. Registracija

Registracija prema svojoj definiciji predstavlja pojam unošenja podataka u registar („Hrvatski jezični portal“, bez dat.), a u digitalnom svijetu to znači unošenje osnovnih podataka u aplikaciju, preciznije bazu podataka, putem kojih će se korisnik dalje moći identificirati. Najčešći pristup registraciji se provodi tako da korisnik unese svoj email i lozinku te opcionalno i dodatne podatke poput imena, prezimena, dobi, broja telefona i slično. Kada korisnik unese tražene podatke koji prođu osnovnu validaciju oni se spremaju u bazu podataka te aplikacije. Važno je naglasiti kako je ovaj proces nužno provesti uz odgovarajuće mjere zaštite. Primjerice standardna praksa je sažimanje (eng. *hashing*) i soljenje (eng. *salting*) lozinki (Anderson, 2008, Poglavlje 2) prije nego se iste spreme u bazu podataka kako bi se u slučaju napada smanjila mogućnost kompromitacije korisničkih računa.

Uz standardni pristup registraciji postoji i manje korišten pristup koji se inicijalno odvija samo putem email adrese. U ovom pristupu korisnik prvo izvrši upit za registraciju putem određene email adrese te nakon toga na njegovu email adresu stiže potvrda o zahtjevu uz koju se nalazi ili poveznica putem koje će se korisnikov zahtjev autentificirati ili privremena lozinka koju će korisnik morati promijeniti kada se jednom prijavi u aplikaciju. Ovaj pristup puno je manje korišten, posebno kada je riječ o mobilnim aplikacijama.

O tome kako se registracija implementira kada je riječ o Android aplikaciji biti će govoreno u zasebnom poglavlju o implementaciji.

3.2. Prijava

Prijava predstavlja proces zaštite imovine i funkcionalnosti sustava od neovlaštenog pristupa te kako Schneier navodi: „Nema smisla štititi pristup imovini ako joj pokušava pristupiti njen valjani vlasnik“ (Schneier, 2006, Poglavlje 13). Prema tome ovaj proces može se podijeliti na tri glavna koraka: identifikaciju, autentifikaciju i autorizaciju. Svaki od koraka nadovezuje se

jedan na drugoga, a kroz ovo poglavlje uz razradu koraka objasnit će se i raznovrsni pristupi svakom od koraka.

3.2.1. Identifikacija

Identifikacija odgovara na pitanje tko pristupa sustavu (Schneier, 2006, Poglavlje 13). Dakle cilj procesa je utvrditi identitet korisnika, ne ga nužno i potvrditi. Kao praktičan primjer možemo razmotriti kupovinu studentske karte za vlak. Prvi korak u svemu tome je identifikacija osobe kao studenta. Da bi osoba koja pokušava kupiti studentsku kartu to i mogla napraviti prvenstveno mora biti student, što će dokazati svojim matičnim brojem studenta koji se nalazi na studentskoj iskaznici. Tu korak identifikacije staje, jer ako osoba ima studentsku iskaznicu to znači da je dala tražene podatke koji se dalje mogu autentificirati.

Slično tome funkcionira i identifikacija podataka u mobilnim aplikacijama. Ako se želimo prijaviti u aplikaciju koja od nas zahtjeva email i lozinku, identifikacija se obavlja kada korisnik unese te podatke. Identifikacija samo potvrđuje da su ti podaci valjani, odnosno da je zadovoljena određena forma u kojoj email i lozinka moraju biti predani na autentifikaciju, ona ne potvrđuje da su podaci autentični ili da njihova kombinacija zaista odgovara korisničkom računu koji se nalazi u bazi podataka aplikacije.

Na primjer ako sustav zahtjeva da unesemo brojčanu vrijednost od šest znamenaka, a mi unesemo slova identifikacija parametara neće proći i samim time nikada neće doći do autentifikacije jer nije zadovoljena osnovna forma u kojoj se proslijeđeni parametri moraju nalaziti.

3.2.2. Autentifikacija

Nakon što identifikacija da potvrdan rezultat dolazi do autentifikacije. Autentifikacija je proces koji potvrđuje identitet korisnika koji pristupa sustavu (McDonald, 2020, Poglavlje 11), odnosno njegova glavna zadaća je da dokaže da su podaci koji su prošli identifikaciju valjani (Schneier, 2006, Poglavlje 13). Ako nastavimo sa primjerom kupovine studentske karte za vlak, autentifikacija podataka zapravo osigurava da osoba koja želi kupiti kartu uistinu je student i to onaj student za kojeg se predstavlja. Dakle, provjerava se matični broj studenta i ako on odgovara osobi koja podnosi zahtjev, primjerice po imenu i prezimenu, onda znamo da ta osoba zaista je student te na osnovu toga možemo prijeći na autorizaciju. Ako u ovoj fazi ne možemo sa sigurnošću potvrditi da se radi o studentu ili da osoba koja je student je ista osoba kao i ona koja izvršava zahtjev za kupnju karte, onda tu proces staje.

U mobilnim aplikacijama proces je opet sličan. Uzimajući ponovno primjer iz prijašnjeg poglavlja sa email adresom i lozinkom u procesu autentifikacije aplikacija će provjeriti postoji li korisnik sa zadanom email adresom te ako postoji provjerit će podudara li se unešena lozinka

sa onom koja se nalazi u bazi. Ako postoji korisnik sa danom email adresom i lozinke se podudaraju aplikacija će prijeći na proces autorizacije. Ako korisnik ne postoji ili se lozinke ne podudaraju proces se tu prekida te je standardna praksa da se korisnika obavijesti o krivo unesenim podacima.

Međutim u mobilnim aplikacijama autentifikacija putem email adrese i lozinke nije jedina mogućnost. Najčešća je, uz varijaciju gdje se kombiniraju korisničko ime i lozinka, ali postoje mogućnosti autentifikacije i putem biometrijskih podataka, višerazinske autentifikacije ili korištenjem rješenja trećih strana. Ove metode autentifikacije opisane su u zasebnim poglavljima.

3.2.2.1. Autentifikacija putem biometrijskih podataka

Autentifikacija putem biometrijskih podataka danas je sve popularnija jer je i sama biometrijska oprema sve rasprostranjenija i dostupnija. Tako danas standardni mobilni uređaj ima barem jednu vrstu biometrijske autentifikacije, bilo to otisak prsta, skeniranje šarenice oka ili cijelog lica. Među ovim kategorijama najpopularniji oblik biometrijske autentifikacije je putem otiska prsta, zatim skeniranje šarenice a onda i skeniranje cijelog lica (Clarke, Furnell, & Reynolds, 2002, str. 65).

Autentifikacija otiskom prsta ujedno je i jedna od najstarijih metoda kada se govori o biometrijskoj autentifikaciji, a Prlenda u svom završnom radu objašnjava njegovu funkcionalnost ovako: „Metodom analize pojedinosti analiziraju se relativni položaji individualnih karakteristika otiska prsta kao što su: završeci grebena, bifurkacije, uzorak izbočina i udubljenja na površini jagodice prsta…“. Ova metoda može se izvršavati putem optičkih, silicijskih ili ultrazvučnih čitača (Prlenda, 2018, str. 13). Iako je tehnologija raznovrsna, princip je isti. Jednom kada se otisak prsta pohrani na uređaju aplikacija komunicira sa aplikacijskim programskim sučeljem (eng. *application programming interface, API*) kako bi potvrdila valjanost otiska i na osnovu rezultata dalje se vrši autorizacija i generalno logika za prijavu u aplikaciju. Dakle aplikacije ne vrše pohranjivanje podataka o biometriji korisnika niti bilo kakvu obradu istih, već to radi sam mobilni uređaj koji isključivo daje informaciju o tome je li otisak prsta valjan onome koji je već pohranjen na uređaju ili ne.

Kada govorimo o sigurnosti ovog pristupa možemo ga smatrati poprilično sigurnim. Najveća mana lozinki jesu ljudi, odnosno činjenica da ih zaboravljaju. Biometriju je nemoguće zaboraviti ili izgubiti, ali ju je moguće krivotvoriti. Međutim krivotvorenje je teško, a napade je teško generirati (Prlenda, 2018, str. 20) pa je ovaj pristup sve rasprostranjeniji i popularniji danas.

Prema istom principu rade i ostale metode biometrijske autentifikacije uz ključnu razliku da koriste drugačije senzore sa autentifikaciju.

3.2.2.2. Višerazinska autentifikacija

Višerazinska autentifikacija je pristup u kojem se autentifikacija korisnika odvija više od jednom, najčešće dva puta što se naziva dvorazinska autentifikacija. To znači da prilikom prijave u aplikaciju nije dovoljno da korisnik upiše točne podatke poput email adrese i lozinke već se mora dodatno autentificirati i putem druge metode autentifikacije. Postoje različiti pristupi višerazinske autentifikacije, a dva najpopularnija su putem SMS-a na mobitelu i korištenjem jednokratne lozinke (eng. *one time password, OTP*) koju generira zasebna aplikacija.

Metoda putem jednokratnog kôda poslanog na broj mobitela u obliku SMS poruke korisnika jedna je od najčešćih metoda višerazinske autentifikacije. Ovim putem također se često dobivaju kôdovi za potvrdu registracije prilikom kreiranja korisničkih računa. Iako je metoda jako popularna nije vrlo pouzdana jer ne zahtijeva da korisnik ima pristup svom uređaju, već svojoj SIM kartici na kojoj je mobilni broj na kojeg se šalje SMS poruka (STANISLAV, 2015, Poglavlje 4). Upravo zbog ovoga ova metoda podložna je brojnim vrstama napada. Međutim popularna je upravo zbog lakoće implementacije u živote korisnika.

Autentifikacija putem jednokratnih lozinki pristup je kojim se generiraju kratke privremene lozinke koje služe kao dodatna provjera nakon inicijalno uspješne provjere identiteta korisnika. Dakle, iako korisnik prođe prvu autentifikaciju putem primjerice korisničkog imena i lozinke bit će zatražen da unese i privremenu jednokratnu lozinku kojom će dodatno potvrditi da se radi o valjanom vlasniku korisničkog računa kojem pokušava pristupiti.

Jednokratne lozinke možemo podijeliti u dvije skupine, one koje su bazirane na vremenu (eng. *time-based one-time password* – TOTP) i one koje su bazirane na sažetku (eng. *hash-based one-time password* – HOTP). Glavna prednost ovog pristupa je kontinuirana promjena lozinke prilikom svake prijave što značajno smanjuje rizik od zlouporabe (Kermek, Novak, Kaniški, & Levak, 2022, str. 45). Ovaj pristup je siguran zbog potrebe za eksternom aplikacijom, poput Google Authenticatora, koja generira privremenu lozinku. Tako se osigurava da čak i ako dođe do kompromitiranja osnovnih podataka za prijavu, korisnički račun ostane siguran jer mu se ne može pristupiti bez dodatne provjere putem jednokratne lozinke.

Implementacija višerazinske autentifikacije bit će prikazana u poglavlju o implementaciji.

3.2.2.3. Autentifikacija putem trećih strana

Kako McDonald u svojoj knjizi navodi: „Najsigurniji autentifikacijski sustav je onaj koji ne moraš pisati sam“ (McDonald, 2020, Poglavlje 9). Danas je ovaj pristup sve popularniji jer pruža lakše rješenje programerima i korisnicima za obavljanje prijave, standardnog procesa u većini aplikacija.

Korisnicima ovo ide na ruku jer se sa jednim računom, primjerice Facebook računom, mogu prijaviti u više aplikacija što znači da za više aplikacija moraju zapamtiti samo jednu kombinaciju lozinke i email adrese. S druge strane ovaj pristup odgovara i programerima jer ne moraju implementirati razne mjere sigurnosti i sigurnosne protokole kako bi osigurali korisničke podatke već se oslanjaju na treće strane da to učine za njih. Glavni nedostatak ovog pristupa je ovisnost o trećim stranama pa je važno da se aplikacije oslanjaju na velika tehnološka poduzeća poput Google-a, GitHub-a ili Meta-e (prije poznate kao Facebook) za ovaj proces.

Ovaj pristup baziran je na Security Assertion Markup Language (SAML) protokolu koji omogućuje implementaciju Single Sign-On (SSO) tehnologija za autentifikaciju tako da mu omogući pristup resursima ako treća strana potvrdi njegovu autorizaciju (Mehta, 2014, Poglavlje 3). Odnosno, ako aplikacija korisni SAML za autentifikaciju putem Facebook računa, nakon što se autentificira pristup ta ista aplikacija mora autorizirati korisnika. S druge strane OpenID Connect protokol (OIDC) je autentifikacijski protokol izgrađen kao dodatak na Open Authorization protokol (OAuth) koji se također koristi za implementaciju Single Sign-On tehnologija te je jednostavniji i u kombinaciji s OAuth protokolom predstavlja budućnost implementacije SSO tehnologija u aplikacije (Mehta, 2014, Poglavlje 3).

Međutim danas postoje i centralizirana gotova rješenja trećih strana koja omogućavaju korištenje više SSO tehnologija baziranih na OAuth protokolu bez potrebe da ih programer sam implementira. Neki od davatelja ovih usluga jesu Okta (Auth0), OneLogin te Centrify (McDonald, 2020, Poglavlje 9).

Više o tome kako ovi protokoli rade bit će objašnjeno u zasebnim poglavljima o njima, a implementacija te će se u poglavlju o implementaciji prikazati implementacije jednog od centraliziranih rješenja.

3.2.3. Autorizacija

Autorizacija predstavlja zadnji korak u procesu prijave u aplikaciju. Prema McDonaldu autorizacija je proces u kojem se odlučuje koje funkcionalnosti korisnik može ili ne može izvršiti (McDonald, 2020, Poglavlje 11). Gledamo li na autorizaciju kao na provjeru prava prilikom podnošenja zahtjeva za izvršavanjem funkcionalnosti možemo reći da je autorizacija proces u kojem se provjerava ima li podnositelj zahtjeva pravo za izvršenje istog (McDonald, 2020, Poglavlje 11; Mehta, 2014, Poglavlje 3). Nadovežemo li se na primjer od prije, jednom kada smo potvrdili da je kupac studentske karte za vlak uistinu student za kojeg se predstavlja možemo mu prodati kartu, a na toj karti pisat će broj sjedala i broj reda u kojem će student sjediti. Tako smo dakle utvrdili identitet korisnika, potvrdili da je on zaista onaj za kojeg se

predstavlja i ograničili smo mu pristup prema njegovoj ulozi što znači da on ima pravo sjediti na zadanom sjedalu u zadanom redu.

Na isti ovaj način autorizacija funkcionira i u mobilnim aplikacijama. Jednom kada smo potvrdili identitet korisnika ograničavamo njegov pristup. Često se to radi prema klasifikaciji u koju korisnik spada. Primjerice ako promatramo Spotify, to je aplikacija koja ima svoje „Premium“ i „Free“ korisnike. Ako korisnik spada pod „Free“ klasifikaciju što se utvrđuje autentifikacijom, u procesu autorizacije bit će mu ograničene funkcionalnosti na one koje su dostupne toj klasifikaciji. Dakle korisnik neće moći slušati pjesme dok nema interneta jer ta funkcionalnost nije dostupna unutar njegove klasifikacije. Tehnički gledano jednostavno će biti odbijen kada proba pristupiti toj funkcionalnosti zbog nevaljanih prava pristupa, ali će moći koristiti sve funkcionalnosti koje se nalaze unutar opsega njegove klasifikacije.

Današnji standard u provođenju sigurne autorizacije je OAuth 2.0 protokol, a više o tome kako protokol radi govorit će se u zasebnom poglavlju o protokolu.

4. Protokoli za autentifikaciju i autorizaciju

Protokol kao pojam u računarstvu predstavlja standardizirani način komunikacije između uređaja za razmjenu podataka. Kao što je već spomenuto u prethodnim poglavljima kako bi valjano implementirali siguran pristup aplikaciji važno je da koristimo industrijske standarde u obliku sigurnosnih protokola.

U ovom poglavlju obradit će se već spomenuti OAuth, OIDC i SAML protokoli, a posebna pažnja dati će se OAuth protokolu, konkretno 2.0 verziji istog budući da je ona današnji standard u izradi aplikacija.

4.1. Open Authorization 2.0 (OAuth 2.0)

Prema svojoj definiciji OAuth protokol je protokol kojim korisnik autorizira aplikaciju za pristup njegovim korisničkim podacima bez da joj direktno da svoj email i lozinku (Mehta, 2014, Poglavlje 3), odnosno svoje vjerodajnice (eng. *credentials*) za pristup određenim podacima. Drugim riječima, omogućava siguran i ograničen pristup resursima na poslužitelju u ime korisnika. U nastavku govoriti će se o OAuth 2.0 protokolu, novijoj verziji koja je današnji standard u razmjeni podataka prilikom korištenja trećih strana. Kroz nadolazeća poglavlja pokušati će se objasniti rad protokola, tok komunikacija u različitim pristupima, razlike između standardne implementacije i one na mobilnim uređajima kao i specifične informacije vezane uz protokol kao što su uloge i tokeni.

4.1.1. Zašto koristiti OAuth

Prema tradicionalnom modelu autentifikacije, klijent pristupa zaštićenim resursima na poslužitelju tako što mu pruža svoje vjerodajnice (korisničko ime i lozinku). Kako bi aplikacije trećih strana imale pristup zaštićenim resursima klijent njima mora dati svoje vjerodajnice. U ovoj analogiji važno je naglasiti da je klijent ujedno i vlasnik tih resursa odnosno da im ima pravo pristupa. Taj model ima sljedeće probleme kada govorimo o komunikaciji s aplikacijama trećih strana:

- **Pohrana podataka na aplikacijama** - aplikacije moraju pohranjivati korisničke podatke za autentifikaciju
- **Autentifikacija putem lozinki** – poslužitelj mora podržavati autentifikaciju putem lozinki unatoč rizicima koji postoje

- **Ograničavanje pristupa** – korisnici ne mogu ograničiti pristup pojedinim resursima već aplikacije treće strane automatski imaju pristup svim resursima koji su dostupni
- **Poništavanje pristupa** – korisnici ne mogu jednostavno poništiti pravo pristupa jednoj aplikaciji treće strane bez mijenjanja lozinke za sve aplikacije što automatski poništava pristup svim aplikacijama
- **Kompromitacija podataka** – ako dođe do kompromitacije aplikacije treće strane dolazi i do kompromitacije korisnika.

Ukratko, u tradicionalnom pristupu korisnik dijeli svoje pristupne podatke s aplikacijama trećih strana što stvara dodatne sigurnosne rizike.

OAuth ovaj problem rješava uvođenjem dodatnog sloja zaštite, autorizacije, kojim razdvaja klijenta i vlasnika resursa. Umjesto da koristi vjerodajnice vlasnika resursa, OAuth koristi pristupne tokene (eng. *access token*). Ovi tokeni imaju određene pristupne atribute između kojih su i doseg te vijek trajanja tokena. Tokeni se izdaju aplikacijama trećih strana putem autorizacijskog poslužitelja uz dopuštenje vlasnika resursa. Klijent zatim koristi taj token za pristup zaštićenim resursima na poslužitelju (Hardt, 2012, Poglavlje 1).

U praktičnom pogledu, OAuth preusmjerava provjeru autentičnosti na poslužitelj koji upravlja korisničkim računom. Taj poslužitelj vraća autorizacijski token aplikaciji treće strane, omogućujući pristup zaštićenim resursima bez direktnog dijeljenja korisničkih vjerodajnica. Ovaj pristup može se usporediti s biometrijskom autentifikacijom na mobitelu. Kao što je spomenuto u poglavlju o biometrijskoj autentifikaciji, aplikacije koje koriste biometrijsku autentifikaciju putem otiska prsta nemaju pristup stvarnim biometrijskim podacima. Umjesto toga, uređaj vrši autentifikaciju i aplikaciji pruža povratnu informaciju o uspješnosti, omogućujući daljnju autorizaciju bez izlaganja osjetljivih podataka. Slično tome koristeći OAuth protokol korisnik zapravo koristi pristupni token za pristup sadržaju, a ne svoje vjerodajnice.

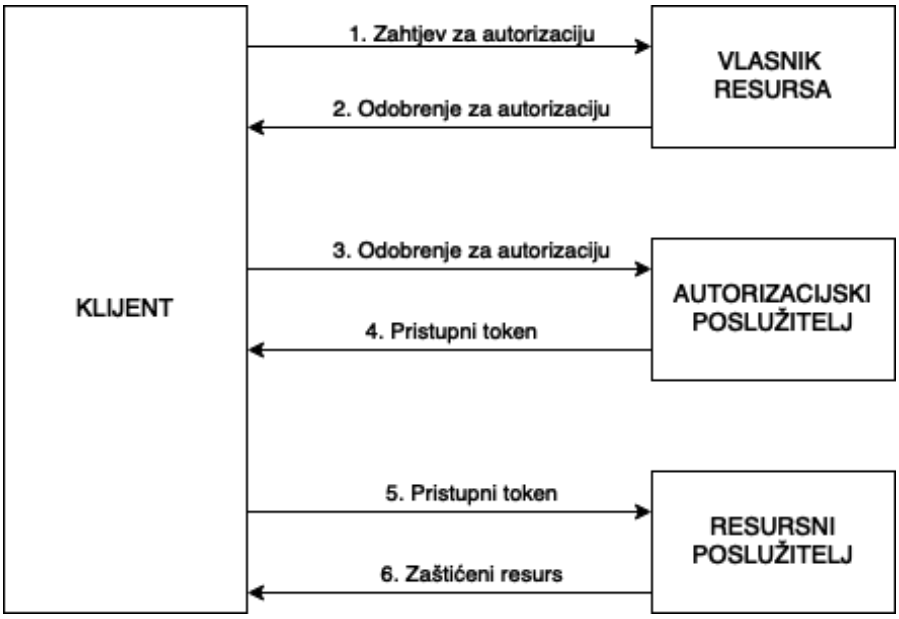
Dakle, OAuth koristimo jer pruža veću razinu sigurnosti prilikom provođena prijave u aplikaciju putem aplikacija trećih strana kao što su prijave koristeći Google, Facebook ili GitHub račune.

4.1.2. Uloge u OAuth-u i njihova komunikacija

OAuth definira četiri uloge (eng. *roles*): vlasnik resursa (eng. *resource owner*), resursni poslužitelj (eng. *resource server*), klijent (eng. *client*) i autorizacijski poslužitelj (eng. *authorization server*) (Hardt, 2012, Poglavlje 1.1). Vlasnik resursa je entitet koji može davati pravo pristupa zaštićenom resursu te ukoliko je vlasnik resursa osoba onda se naziva krajnji korisnik (eng. *end user*). Resursni poslužitelj je poslužitelj na kojem se resursi nalaze, a koji može primati i odgovarati na zahtjeve koristeći pristupne tokene. Klijent je aplikacija koja vrši zahtjeve za pristup zaštićenim resursima u ime klijenta sa svojom autorizacijom. Na posljetku,

autorizacijski poslužitelj je poslužitelj koji izdaje pristupne tokene nakon uspješne autentifikacije vlasnika resursa i dobivanja autorizacije (Hardt, 2012, Poglavlje 1.1).

Kako bi ove četiri uloge komunicirale, OAuth u svojoj osnovnoj implementaciji ima šest koraka komunikacije. Na slici broj jedan ovi koraci prikazani su dijagramu toka.



Slika 1: Prikaz koraka komunikacije OAuth protokola tijekom autorizacije (Hardt, 2012, Poglavlje 1.2)

Ovdje vidimo kako se putem klijenta obavljaju svi koraci odnosno zahtjevi kako bi se na kraju dobio pristup zaštićenim resursima. Prvo, klijent traži autorizaciju od vlasnika resursa. Ovaj zahtjev može se poslati direktno vlasniku odnosno krajnjem korisniku kao što je prikazano na slici ili autorizacijskom poslužitelju kao posredniku što je i preferirana metoda. Dalje klijent prima odobrenje za autorizaciju (eng. *authorization grant*) koju prema specifikaciji OAuth protokola može dobiti na četiri načina. Više o samom odobrenje za autorizaciju bit će govoreno u sljedećem poglavlju. Kada ima odobrenje za autorizaciju klijent ga šalje u zahtjevu prema autorizacijskom poslužitelju od kojeg traži pristupni token (eng. *access token*). Autorizacijski poslužitelj zatim autorizira klijenta i provjerava autorizacijski tip potpore te ukoliko prođe provjeru vraća mu pristupni token. Koristeći pristupni token klijent sada traži od resursnog poslužitelja zaštićene resurse te ukoliko je taj token ispravan i prođe provjeru resursnog poslužitelja on će klijentu vratiti zaštićene resurse.

4.1.3. Načini dobivanja odobrenja za autorizaciju

Kao što je u prethodnom poglavlju spomenuto, odobrenje za autorizaciju je oblik vjerodajnice koji predstavlja autorizaciju vlasnika resursa te on služi za dobivanje pristupnog tokena od autorizacijskog poslužitelja. Za dobivanje ovog odobrenja specificirana su četiri

načina: putem autorizacijskog kôda (eng. *authorization code*), implicitnim putem (eng. *implicit*), korištenjem vjerodajnica vlasnika resursa (eng. *resource owner password credentials*) te korištenjem vjerodajnica klijenta (eng. *client credentials grant*) (Hardt, 2012, Poglavlje 1.3). Svaki od ovih načina bit će ukratko objašnjen u svom potpoglavlju.

4.1.3.1. Način korištenjem autorizacijskog kôda

Ovo je preferirana metoda autorizacije te je i najčešća, a pogodna je za web aplikacije. U ovom načinu, autorizacijski kôd dobiva se tako da se zahtjev šalje direktno autorizacijskom poslužitelju, koji služi kao posrednik između klijenta i vlasnika resursa. Klijent preusmjerava vlasnika resursa na autorizacijski poslužitelj putem web preglednika, gdje se vlasnik resursa autentificira. Nakon uspješne autentifikacije i autorizacije, autorizacijski poslužitelj vraća vlasnika resursa nazad na klijenta s autorizacijskim kôdom.

Kada klijent ima autorizacijski kôd isti taj kôd koristi kako bi poslao još jedan zahtjev na autorizacijskog poslužitelja u kojem traži pristupni token. Ukoliko je kôd valjan autorizacijski poslužitelj vratiti će pristupni token te se pristupni token dalje koristi za pristup resursima. Klijent tada koristi pristupni token za daljnji pristup resursima.

Da sumiram, u ovom načinu vlasnik resursa autentificira se samo na autorizacijskom poslužitelju što znači da klijent nikad ne vidi njegove vjerodajnice. Ovo je glavna sigurnosna prednost ovog procesa.

4.1.3.2. Implicitni način

Implicitni način je pojednostavljeni pristup koji se koristi kod javnih klijenata, gdje se pristupni token vraća odmah u URI fragmentu, izbjegavajući dodatni korak zamjene autorizacijskog kôda za pristupni token kao što je to bilo u pristupu korištenjem autorizacijskog kôda. Ovaj pristup naziva se implicitnim jer se ne koristi nikakav međukorak poput autorizacijskog kôda, već se token vraća direktno u web preglednik (Hardt, 2012, Poglavlje 1.3).

Ukratko, ovaj način funkcionira tako da klijent šalje svoju identifikaciju i povratni URI (eng. *redirect URI*) na autorizacijski poslužitelj kako bi poslužitelj znao gdje vratiti pristupni token. Nakon što autorizacijski poslužitelj autentificira korisnika, preusmjerava preglednik natrag na povratni URI s pristupnim tokenom u URI fragmentu. Klijent zatim preuzima taj token iz URI fragmenta i koristi ga za pristup zaštićenim resursima na web pregledniku.

Implicitni način inicijalno je dizajniran za javne klijentske aplikacije koje se izvršavaju u web pregledniku, poput jednostraničnih aplikacija (eng. *single page application – SPA*) te prema efikasnosti može biti brži od ostalih načina. Međutim njegova uporaba se ne preporuča budući da tokeni putuju kroz web preglednik što ih potencijalno izlaže napadima (Hardt, 2012, Poglavlje 10).

Ovaj način ima određene sličnosti sa implementacijom OAuth protokola u izvornim aplikacijama (eng. *native applications*) kao što su i mobilne aplikacije no bez dodatnih sigurnosnih mjera koje one poduzimaju, ali više o OAuth-u u mobilnim aplikacijama biti će govoreno u poglavlju u zasebnom potpoglavlju.

4.1.3.3. Način korištenjem vjerodajnica vlasnika resursa

U ovom pristupu kako bi se dobio pristupni token koriste se vjerodajnice vlasnika resursa poput njegovog korisničkog imena i lozinke. Iako može izgledati nesigurno ovaj način u teoriji eliminira potrebu da klijent pohranjuje vjerodajnice vlasnika uporabom pristupnih tokena dužeg životnog vijeka ili uporabom osvježavajućeg tokena (eng. *refresh token*) (Hardt, 2012, Poglavlje 1.3).

Ova metoda se koristi kada klijent ima povjerenje vlasnika resursa i kada druge metode nisu prikladne. Klijent izravno prikuplja vjerodajnice korisnika i šalje ih autorizacijskom poslužitelju uz zahtjev za pristupni token. Autorizacijski poslužitelj tada provjerava vjerodajnice, autentificira vlasnika resursa i vraća pristupni token klijentu.

Ovaj pristup ipak se ne preporučuje za generalnu uporabu zbog sigurnosnih razloga. Korištenje vjerodajnica vlasnika resursa povećava rizik od kompromitacije jer vjerodajnice prolaze kroz klijent i mogu biti pohranjene ili presretnute. Zbog toga se ovaj način koristi samo u posebnim slučajevima.

Ovaj način može biti prikladan kada se radi o aplikacijama visokog povjerenja kao što su službene aplikacije naše organizacije, ali se ne preporučuje za aplikacije trećih strana ili javne aplikacije zbog povećanih sigurnosnih rizika.

4.1.3.4. Način korištenjem vjerodajnica klijenta

Ovaj način koristi vjerodajnice klijenta kao odobrenje za autorizaciju (Hardt, 2012, Poglavlje 1.3). Dakle klijent direktno autentificira sebe autorizacijskom poslužitelju koristeći svoje vjerodajnice te mu autorizacijski poslužitelj daje pristupni token.

Ovaj pristup pogodan je kada klijent treba pristupiti vlastitim resursima ili resursima kojima kontrolira, odnosno kada je klijent ujedno i vlasnik resursa. Primjerice ovaj način koristan je kod komunikacije s API-ima koji ne zahtijevaju interakciju s krajnjim korisnikom ili kod međusobne komunikacije poslužitelja.

Ipak ovaj pristup ne bi trebao biti korišten u drugim slučajevima, posebno kada se vrši komunikacija s krajnjim korisnikom ili s aplikacijama trećih strana.

4.1.4. Pristupni token

Kroz poglavlja već je puno bilo govora o pristupnim tokenima, a ukratko oni predstavljaju apstrakciju vjerodajnica kojima korisnik pristupa zaštićenim resursima (Hardt,

2012, Poglavlje 1.4). Dakle, umjesto da klijent pristupa resursnom poslužitelju koristeći korisničko ime i lozinku, koristi pristupni token.

Tokeni su obično nečitki klijentima, a u sebi sadržavaju atribute koji definiraju doseg i vijek trajanja pristupa. Taj sadržaj čita resursni poslužitelj, a on se može mijenjati ovisno o propisanim potrebama istog tog poslužitelja.

Ovi tokeni još se nazivaju „bearer“ tokeni prema RFC6750 standardu (Jones & Hardt, 2012, Poglavlje 1.2). Prema istom standardu definirana su tri načina slanja ovih tokena kada govorimo o slanju zahtjeva za pristup zaštićenim resursima. Sva tri načina šalju se putem HyperText Transfer protokola (HTTP), a dijele se na slanje u zaglavlju zahtjeva, tijelu zahtjeva te kao parametar u URI-u. Važno je napomenuti kako se klijent treba odlučiti samo za jednu metodu koju će dalje nastaviti koristiti. U daljnjim potpoglavljima objasniti će se zasebne metode slanja kao i atributi koji se nalaze u tokenima.

4.1.4.1. Metode slanja Bearer tokena

Slanje tokena u zaglavlju (eng. *Authorization header*) je najčešći i preporučen pristup te resursni poslužitelji moraju podržavati ovu metodu zaprimanja tokena. U ovom načinu autorizacijski dio zaglavlja izgledao bi ovako: „Authorization: Bearer token“. Authorization je parametar unutar zaglavlja, bearer je riječ koja se dodaje uz token kako bi se naznačilo da se radi o pristupnom tokenu, a token predstavlja sam token koji šaljemo, a koji je sam u obliku niza (eng. *string*).

Sljedeći način je slanjem tokena u tijelu (eng. *Form-encoded Body Parameter*) gdje se tijelu zahtjeva dodaje ključna riječ „access_token“. Korištenje ovog pristupa zahtjeva nekoliko pravila oko postavljanja samog zahtjeva kao što su postavljanje „Content-Type“ vrijednosti u zaglavlju zahtjeva parametra na „application/x-www-form-urlencoded“, sadržaj tokena mora biti zapisan isključivo u ASCII znakovima, ne smije biti korištena „GET“ metoda zahtjeva te tijelo zahtjeva mora pratiti HTML 4.01 specifikaciju (Jones & Hardt, 2012, Poglavlje 2.2). Iako se ovaj način ne preporuča ipak svoju primjenu može naći kada web preglednik ne može pristupiti zaglavlju zahtjeva pa je forsiran koristiti tijelo.

Zadnji način slanja zahtjeva, ujedno i najnesigurniji, je putem URI parametara (eng. *URI Query Parameter*). Ovdje se na URI korišten za pristup zaštićenim resursima dodaje „?access_token=token“ parametar te bi primjer cijelog URI-a onda izgledao ovako: https://www.example.com/resource?access_token=token, a zbog izloženosti tokena i laganog praćenja i pohranjivanja URI-a se metoda i ne preporučuje.

4.1.4.2. Atributi u Bearer tokenu

Neovisno o metodi koju smo odabrali za slanje tokena, svaki token ima svoje atribute. U odgovoru uspješnog zahtjeva za dobivanje pristupnog tokena dobit ćemo odgovor koji u sebi

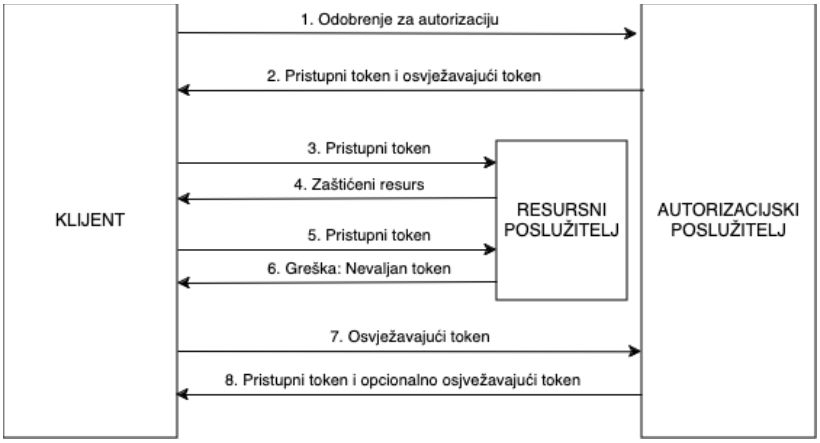
mora sadržavati pristupni token (eng. *access_token*) i njegov tip (eng. *token_type*), a opcionalno može još sadržavati vijek trajanja tokena (eng. *expires_in*) ukoliko on može isteći, doseg tokena (eng. *scope*) ukoliko je doseg različit od onoga koji je zahtjevan te opcionalno još može sadržavati i osvježavajući token (eng. *refresh_token*) (Jones & Hardt, 2012, Poglavlje 3). O tome će biti više govora kasnije. Također preporuča se koristiti statusni kôd 200 u slučaju uspješnog odgovora.

U slučaju neuspješnog zahtjeva preporuča se da se vraćaju odgovori sa statusnim kôdom 400 sa parametrima greške (eng. *error*), a opcionalno mogu se vratiti i opis greške (eng. *error_description*) kao i URI za bolje razumijevanje greške (eng. *error_uri*). Greška će se uvijek vratiti u jednom od mogućih odgovora: nevaljan zahtjev (eng. *invalid_request*) kada primjerice fali neki od atributa u zahtjevu, nevaljan token (eng. *invalid_token*) kada je token istekao, krivo je formatiran i slično te nedozvoljen doseg (eng. *insufficient_scope*) kada iako token je valjan on nema pravo pristupa zbog ograničenja u dosegu. U slučaju nedozvoljenog dosega preporuča se da se vrati statusni kôd 403 kako bi se naznačilo da je pristup zabranjen (Jones & Hardt, 2012, Poglavlje 6).

U oba slučaja odgovori se vraćaju u JSON obliku, a važno je napomenuti kako kada se vraća uspješni odgovor mora se naglasiti „no-store“ vrijednost u „Cache-Control“ parametru odgovora kako ne bi došlo do pamćenja krivih ili isteknutih tokena.

4.1.5. Osvježavajući token

Osvježavajući token predstavlja vjerodajnicu za dobivanje pristupnog tokena (Hardt, 2012, Poglavlje 1.5). Osvježavajući token vraća se zajedno sa inicijalnim pristupnim tokenom te to radi autorizacijski poslužitelj kao što je već bilo govora u prethodnom poglavlju, a trebaju se koristiti onda kada trenutni pristupni token istekne ili izgubi prava pristupa, primjerice prema svom dosegu. Ovaj proces vidljiv je i na slici dva.



Slika 2: Prikaz toka tijekom korištenja pristupnog i osvježavajućeg tokena (Hardt, 2012, Poglavlje 1.5)

Kako je vidljivo na slici, za razliku od pristupnog tokena, osvježavajući token koristi se isključivo kako bi se poslao zahtjev autorizacijskom poslužitelju za izdavanje novog pristupnog tokena. Osvježavajući token ne može se koristiti za komunikaciju sa resursnim poslužiteljem.

4.1.6. Klijent

Kroz prethodna poglavlja puno je bilo govora o klijentu i kakvu ulogu on predstavlja u protokolu. Međutim da bi klijent, a ujedno i sam protokol, postojao mora postojati i proces registracije klijenta. Nakon registracije definiraju se tip i profil klijenta. Kroz daljnja poglavlja opisati će se ovi pojmovi.

4.1.6.1. Registracija klijenta

Kako se OAuth protokol primjenjuje za integraciju autorizacije pomoću aplikacija trećih strana tako se registracija u ovom smislu odnosi na registraciju klijenta u aplikaciju treće strane koju mi koristimo kao formu autorizacije u našoj aplikaciji. Primjeri takvih aplikacija bili bi Google, Facebook ili GitHub. Tijekom registracije uobičajeno se definira tip klijenta, a često i profil klijenta.

Kako bi bilo lakše pratiti uzeti ću za primjer TO-DO aplikaciju koja predstavlja klijenta i primjer GitHub autorizacije koja će predstavljati autorizacijski poslužitelj koji koristim u njoj. Prvi korak u registraciji klijenta jest kreiranje nove OAuth aplikacije koju je potrebno povezati sa TO-DO aplikacijom, primjerice prema SHA1 sažetku ili nazivu paketa (eng. *package* name) kada se govori o Android aplikacijama. Zatim nam OAuth aplikacija izdaje svoje vjerodajnice odnosno id klijenta (eng. *client id*) i tajnu klijenta (eng. *client secret*). Te vjerodajnice je zatim potrebno uključiti u TO-DO aplikaciju jer će se putem njih komunicirati s autorizacijskim poslužiteljem.

Kada se korisnik odluči prijaviti u TO-DO aplikaciju putem GitHub računa on će koristiti svoje GitHub vjerodajnice tijekom prijave u GitHub preko TO-DO aplikacije, konkretno svoju email adresu i lozinku. Tijekom ovog procesa TO-DO aplikacija komunicira sa GitHub-om koji ovisno o valjanosti vjerodajnica korištenih na klijentu može odobriti ili opovrgnuti proces prijave putem GitHub-a. Ako je proces prijave odobren tijekom te prijave korisnik će morati autorizirati TO-DO aplikaciju, odnosno odobriti dijeljenje svojih podataka sa istom, a ako to učini TO-DO aplikacija moći će ga dalje autorizirati unutar sebe odnosno korisnik će dobiti pristup zaštićenim resursima aplikacije.

Da bi ovaj proces funkcionirao korisnik koji se prijavljuje sa GitHub računom mora već imati svoj GitHub račun te mora odobriti autorizaciju TO-DO aplikacije. Za ovaj pristup značajno je da klijentska aplikacija, TO-DO u našem slučaju, nema nikakve podatke o korisniku koji se prijavio u nju, ali putem pristupnog tokena može potvrditi valjanost njegove prijave.

Više o tome kako ovaj proces izgleda u konkretnom primjeru sa programskim kôdom bit će govoreno u poglavlju o implementaciji.

4.1.6.2. Tip i profil klijenta

U specifikaciji OAuth protokola postoje dva tipa klijenta: povjerljivi (eng. *confidential*) i javni (eng. *public*), a razlikuju se prema svojoj sposobnosti da zaštite svoje vjerodajnice (Hardt, 2012, Poglavlje 2.1).

Povjerljivi klijenti su klijenti koji imaju siguran način zaštite vjerodajnica, a najčešće su to klijenti koji se izvršavaju na poslužitelju, a prema profilu spadaju u web aplikacije.

Javni klijenti su klijenti koji svoje vjerodajnice moraju čuvati na uređaju na kojem se aplikacija izvršava što ih čini nesigurnima. Prema svom profilu ove aplikacija jesu izvorne i jednostranične aplikacije. U ovakvim aplikacijama se primjenjuje implicitni tok ili Proof Key of Code Exchange (PKCE) tok koji služe za podizanje razine sigurnosti budući da ne zahtijevaju tajnu klijenta.

4.1.7. OAuth u mobilnim aplikacijama

U dosadašnjim poglavljima objašnjen je princip rada OAuth protokola prema njegovoj generalnoj specifikaciji. Međutim prema toj specifikaciji mobilne aplikacije spadaju u izvorne aplikacije te kao takve nisu dovoljno sigurne kako bi koristile klasičnu implementaciju OAuth protokola.

Glavni problemi koji se javljaju kod korištenja standardne specifikacije OAuth protokola u mobilnim aplikacijama jesu problem preusmjeravanja koji za sobom povlači i problem nemogućnost potvrde autorizacijskom serveru da šalje token valjanom klijentu te problem da se gotovo uvijek barem dio logike za implementaciju protokola nalazi na klijentu što ga čini ranjivim (Chen i ostali, 2014, Poglavlje 4).

Predloženo rješenje u mobilnim aplikacijama je primjena PKCE protokola, a više o njemu kao i o drugim detaljima i razlikama između mobilne implementacije OAuth protokola i standardne implementacije bit će govoreno u nadolazećim potpoglavljima.

4.1.7.1. PKCE protokol

Kao što je već spominjano kako bi se primijenio OAuth protokol u mobilnim aplikacijama nužno je koristiti PKCE tok podataka. Dva nova koncepta koje ovaj protokol uvodi jesu verifikator kôda (eng. *code verifier*) i izazov kôda (eng. *code challenge*). Proces implementacije sadrži pet koraka (Sakimura, Bradley, & Agarwal, 2015, Poglavlje 4).

Prvi korak kod implementacije ovog protokola je generiranje verifikatora kôda za svaki OAuth zahtjev koji klijent napravi. Verifikator kôda predstavlja kriptirani znakovni niz između 43 i 128 znakova. Naravno ne bi ga trebalo biti moguće pogoditi.

Drugi korak u ovom procesu jest generiranje izazova kôda. To se može napraviti tako da izazov bude jednak verifikatoru, ali ova metoda nije preferirana i trebala bi se provoditi samo ako klijent iz tehničkih razloga ne može koristiti drugu metodu. Druga metoda naziva se „S256“, a izvodi se tako da se verifikator kôda sažima koristeći SHA256 algoritam, a zatim se taj sažetak kodira u Base64 URL formatu.

Treći korak je slanje autorizacijskog zahtjeva. U ovom zahtjevu klijent mora poslati i izazov kôda, a šalje ga uz obavezan atribut „code_challenge“ i opcionalan atribut „code_challenge_method“ koji se odnosi na metodu kojim je izazov kôda dobiven. Moguće vrijednosti za metodu jesu „S256“ ili „plain“ te ako se vrijednost ne pošalje onda autorizacijski poslužitelj pretpostavlja da je „plain“.

U četvrtom koraku poslužitelj vraća autorizacijski kôd klijentu, a ako je zahtjev bio neuspješan onda vraća poruku greške. Važno je naglasiti kako ovdje poslužitelj mora uključiti izazov kôda i njegovu metodu unutar autorizacijskog kôda kako bi se isti mogao potvrditi.

Klijent u zadnjem koraku šalje autorizacijski kôd zajedno s verifikatorom kôda natrag autorizacijskom poslužitelju u zahtjevu za pristupni token. Poslužitelj tada provjerava izazov kôda i verifikator kôda te ako su jednaki vraća pristupni kôd i ostatak implementacije OAuth protokola teče normalno. Ukoliko verifikator i izazov nisu jednaki poslužitelj vraća poruku greške.

Prema ovome vidljivo je zašto je ovo obavezan dio kod korištenja OAuth protokola u mobilnim aplikacijama. Ovaj pristup značajno povećava razinu sigurnosti i onemogućuje dobivanje pristupnog tokena presretanjem autorizacijskog kôda. Čak i kada bi došlo do presretanja napadaču nije poznat verifikator kôda koji je nužan za dobivanje pristupnog tokena. Uz to verifikator kôda je jedinstven za svaki zahtjev što bi onemogućilo druge metode napada.

4.1.7.2. Registracija i autentifikacija mobilnih klijenata

U procesu registracije mobilni klijenti moraju biti naznačeni kao javni klijenti zbog njihove nesigurnosti. Na osnovu te verifikacije autorizacijski poslužitelj tražiti će od klijenta da definira URI-e za preusmjeravanje te će odbijati sve zahtjeve koji nisu upućeni s tih URI-a. Moguće je da će autorizacijski poslužitelji zahtijevati i dodatne atribute poput „app package“ ili „bundle name“ atributa kako bi dodatno mogli potvrditi da se zaista radi o valjanom zahtjevu za autorizaciju.

Proces autentifikacije mobilnog klijenta ne može zahtijevati od njega podatke poput id-a klijenta jer se ne može osigurati njegova tajnost te se ovaj proces onda izvodi prema PKCE toku opisanom u prethodnom poglavlju.

4.2. OpenID Connect (OIDC)

OpenID Connect predstavlja dodatan autentifikacijski sloj izgrađen kao proširenje na OAuth protokol. Glavna svrha OIDC protokola je da omogući klijentskoj aplikaciji dobivanje informacija o identitetu krajnjeg korisnika (Li, 2017, Poglavlje 3.4). Time se omogućava da korisnik dijeli podatke o sebi s klijentskom aplikacijom bez kreiranja zasebnog korisničkog računa za svaku aplikaciju što je i glavna prednost SSO pristupa prijavi. Više o tome kako protokol funkcionira biti će govoreno u nadolazećim poglavljima.

4.2.1. OIDC u suradnji s OAuth protokolom

Implementacija OIDC protokola u suradnji s OAuth protokolom slična je originalnoj implementaciji OAuth protokola uz par značajnih razlika. Ako aplikacija želi koristiti OIDC prilikom autorizacije korisnika u inicijalnom OAuth toku mora još dodati „openid“ atribut kako bi signalizirala autorizacijskom poslužitelju da isti želi koristiti. Dalje proces ostaje gotovo isti. Korisnik se prijavljuje sa svojim vjerodajnicama, dopušta aplikaciji uvid u njegove podatke te klijent dobiva autorizacijski kôd. Kada klijent pošalje autorizacijski kôd nazad autorizacijskom poslužitelju ovisno o njegovoj ispravnosti nazad će uz pristupni token dobiti i novi ID token. Klijent može iskoristiti ID token u daljnjoj komunikaciji s autorizacijskim poslužiteljem kako bi dobio dodatne informacije o korisniku poput njegove email adrese, imena, datuma rođenja, spola i ostalih podataka koje korisnik odluči dijeliti s aplikacijom. Više o samom ID tokenu biti će govoreno u idućem poglavlju.

4.2.1.1. ID token

ID token glavna je inovacija OIDC protokola, a sam token vezan je uz identitet korisnika koji se prijavljuje u aplikaciju. Ovaj token koristi se uz pristupni i autorizacijski token iz OAuth protokola, a sadrži tvrdnje o autentifikaciji korisnika od strane autorizacijskog poslužitelja. Ovisno o potrebi može sadržavati i dodatne tvrdnje, a sam token predstavljen je u JSON Web Token formatu (JWT) te je digitalno potpisan od strane autorizacijskog poslužitelja (Li, 2017, Poglavlje 3.4). Naravno sam token ima određene atribute poput pomoću kojih definira poslužitelja koji je izdao token, identifikator korisnika, identifikator klijenta koji je zatražio token, vrijeme kreiranja i isteka tokena i ostale atribute koji su vidljivi u tablici jedan.

Tablica 1: Prikaz atributa ID tokena (de Medeiros, Agarwal, Sakimura, Bradley, & Jones, 2014, Poglavlje 2)

Atribut	Opis atributa	Obavezan
iss	Identifikator izdavatelja tokena	Da
sub	Identifikator korisnika	Da
aud	Identifikator klijenta	Da

exp	Vrijeme isteka tokena	Da
iat	Vrijeme izdavanja tokena	Da
auth_time	Vrijeme autentifikacije korisnika	Ne
nonce	Vrijednost koja asocira sesiju klijenta i ID token	Preporuča se
acr	Razina autentifikacije koja je korištena	Ne
amr	Metode korištene za autentifikaciju	Ne
azp	Klijent koji je autoriziran za korištenje tokena	Preporuča se

Ovi atributi korisni su za implementaciju OIDC protokola, ali preko njih se ne pristupa informacijama korisnika. Za pristup informacijama o korisniku poput njegovog imena, slike profila, email adrese, spola i ostalih informacija koriste se standardne tvrdnje (eng. *standard claims*). Kako bi do njih došli klijent mora pristupiti „UserInfo“ krajnjoj točci te u zaglavlju mora poslati Bearer token, a u slučaju uspješne provjere klijent će povratno dobiti one informacije o korisniku koje je korisnik dozvolio da se dijele te su naravno iste povezane preko „sub“ atributa u ID tokenu (de Medeiros i ostali, 2014, Poglavlje 5).

4.2.1.2. Sigurnosni aspekti OIDC protokola u suradnji s OAuth protokolom

Kao što je već spomenuto OIDC dodaje dodatni sigurnosni sloj integrirajući ID tokene s OAuth protokolom. Klijenti provjeravaju valjanost tokena kroz nekoliko ključnih koraka kako bi osigurali njegovu autentičnost i integritet.

Prvi korak je provjera digitalnog potpisa. Klijenti koriste javne ključeve koji su javno dostupni putem autorizacijskog poslužitelja za dekripciju tokena i provjeru potpisa. Time se osigurava da token nije mijenjan te da ga je izdao pouzdani izvor. Drugi korak je provjera atributa unutar tokena. Klijent putem tih atributa provjerava osnovne tvrdnje poput valjanosti tokena, izdavatelja tokena, klijenta tokena i slično. Daljnji koraci ovise o specifičnim atributima tokena koji se opcionalno šalju sa njime, a prema svojim zahtjevima mogu zahtijevati detaljniju provjeru.

Ovi koraci provjere ključni su za sigurnost OIDC protokola jer osiguravaju da je token autentičan, nije kompromitiran te da informacije unutar njega odgovaraju stvarnim podacima. Kroz provjeru potpisa, integriteta i atributa tokena, klijentska aplikacija može biti sigurna u identitet korisnika, čime se osigurava sigurnost i povjerenje u cijeli sustav autentifikacije. Ovim pristupom smanjuje se šansa neovlaštenog pristupa aplikaciji te se na taj način dodatno štite i korisnik i aplikacija.

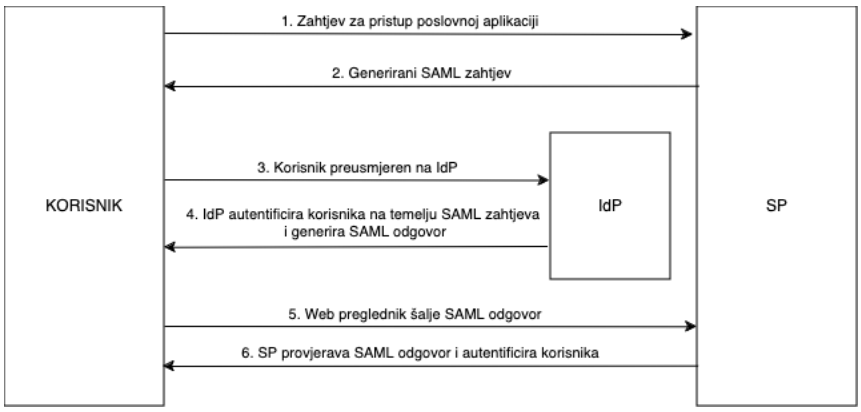
4.3. Security Assertion Markup Language (SAML)

Security Assertion Markup Language ili ukratko SAML standardni je protokol koji omogućuje implementaciju SSO prijave u aplikaciju (Mehta, 2014, Poglavlje 3). SAML je trenutno u svojoj drugoj verziji, a iako sam ga nazvao svojevrsnom pretečom OAuth i OIDC protokola on je i dalje u upotrebi posebno kada se radi o implementaciji SSO-a, posebno za poduzeća. U nadolazećim poglavljima proći ću osnovne principe u radu protokola te ću prikazati razlike između SAML-a i OAuth-a koji kombinira i OIDC protokol.

4.3.1. Principi rada SAML-a

U drugoj iteraciji SAML-a fokus je stavljen na web aplikacije i implementaciju SSO-a na njima te tako su definirane tri uloge: korisnik (eng. *principal*), pružatelj identiteta (eng. *identity provider – IdP*) i pružatelj usluge (eng. *service provider – SP*) (Mehta, 2014, Poglavlje 3). Ove uloge komuniciraju s pomoću XML tipa poruka (Hughes & Maler, 2005, Poglavlje 1). Uloge su slične kao u OAuth-u te SP možemo promatrati kao klijenta, IdP kao autorizacijskog poslužitelja, a korisnika kao krajnjeg korisnika te se može usporediti sa vlasnikom resursa.

Na slici tri vidljiv je sam tok SAML protokola koji ima određene sličnosti sa OAuth tokom.



Slika 3: Prikaz osnovnog SAML toka (Mehta, 2014, Poglavlje 3)

Kako bi bliže opisao SAML tok ponovno ću kao primjer uzeti TO-DO aplikaciju od prije. Ovdje korisnik pokušava pristupiti TO-DO aplikaciji koja na njegov zahtjev generira SAML zahtjev te se korisnik preusmjerava na pružatelja identiteta, recimo GitHub. On obrađuje taj zahtjev te na temelju njega autentificira korisnika i generira SAML odgovor koji možemo promatrati kao pristupni token. Zatim se taj SAML odgovor šalje natrag u TO-DO aplikaciju. Na osnovu SAML odgovora aplikacija autentificira korisnika i dozvoljava mu pristup zaštićenim resursima.

4.3.2. Struktura SAML poruka

Kao što je već spomenuto SAML se oslanja na XML poruke, a one imaju tri osnovna tipa tvrdnji: autentifikacijska tvrdnja (eng. *authentication assertion*), autorizacijska tvrdnja (eng. *authorization assertion*), atributna tvrdnja (eng. *attribute assertion*).

Autentifikacijska tvrdnja sadrže informacije o tome tko je izdao tvrdnju, tko je subjekt koji je autentificiran te koliko dugo autentifikacija traje. Ova tvrdnja omogućuje SP-u da potvrdi identitet korisnika bez potrebe za direktnim pristupom njegovim vjerodajnicama. Atributna tvrdnja odnosi se na podatke koji su vezani uz samog subjekta odnosno korisnika te omogućava SP-u da personalizira iskustvo korisnika. Autorizacijska tvrdnja specificira funkcionalnosti ili resurse kojima subjekt može pristupiti (Hughes & Maler, 2005, Poglavlje 3.2).

Usporedimo li ove tvrdnje s tokenima u OAuth i OIDC protokolima može se primijetiti kako autorizacijska tvrdnja prema svojoj funkcionalnosti sliči pristupnom tokenu, a preostale tvrdnje više odgovaraju ID tokenu.

4.3.3. SAML ili OAuth s OIDC-om

Kao što je vidljivo iz prethodnih poglavlja SAML protokol po mnogočemu je sličan OAuth i OIDC protokolima što je i očekivano budući da definiraju isti način pristupa aplikacijama.

Glavna razlika je u tehnologiji koju protokoli koriste, ali i u fokusu protokola. OAuth i OIDC koriste tokene pisane u JSON formatu dok SAML za formatiranje svojih poruka koristi XML. S jedne strane SAML je fokusiran isključivo na web aplikacije te se uglavnom koristi u korporativnom svijetu dok su OAuth i OIDC više namijenjeni svakodnevnim korisnicima, a i sami protokoli mogu se primijeniti na više vrsta aplikacija.

Dakle ako želimo robusno i sigurno rješenje koje namjeravamo koristiti u korporativnom okruženju baziranom na web aplikacijama te nam njegova kompleksnost nije presudna u odluci naš izbor trebao bi biti SAML protokol. No ako želimo rješenje koje će funkcionirati na više vrsta aplikacija te je prema svom pristupu modernije i lakše za održavanje trebamo razmotriti implementaciju OAuth i OIDC protokola. Naravno ovo nisu jedina pravila za odabir protokola te će krajnja implementacija ovisit o vrsti i zahtjevima projekta ili organizacije.

5. Tehnologije koje podržavaju OAuth

Kroz prethodna poglavlja bilo je puno govora o tome kako je OAuth postao standardom u implementaciji procesa SSO prijave danas te je njegova implementacija široka i raznovrsna kada se govori o platformama koje podržava i kada se govori o tehnologijama sa kojima se može implementirati.

Cilj ovog poglavlja je bliže predstaviti mogućnosti za implementaciju SSO tehnologija u moderne aplikacije te će se tako kroz poglavlje obraditi Firebase Authentication kao danas gotovo standardni pristup razvoju modernih aplikacija za android uređaje, ali će biti i govora o pojedinačnim tehnologijama poput Google Sign-in i Facebook login tehnologija, a zasebno potpoglavlje odvojit će se i za centralizirano rješenje Auth0.

5.1. Firebase Authentication

Firebase Authentication platforma je dizajnirana za olakšavanje procesa osiguravanja pristupa aplikaciji pružajući raznovrsne metode autentifikacije, a zasniva se na prije spomenutim protokolima („Firebase Authentication“, bez dat., od. Introduction). Firebase Authentication samo je dio cijelog skupa Firebase usluga koje podržava Google te se besprijekorno integrira s njima pa je na taj način postala i jednim od standardnih pristupa implementacije autentifikacije u mobilne aplikacije. Uz to, Firebase Authentication nije ograničen samo na Android uređaje. Uz Android platformu nudi podršku i za iOS i Web aplikacije kao i za Unity igre. Ova raznovrsnost pomaže Firebase Authenticationu da se istakne kao sveobuhvatno rješenje za implementaciju autentifikacije.

Kroz ovo poglavlje predstavit ću razne metode koje su dostupne a fokusiraju se na Android operativni sistem te će neke od njih biti implementirane u zasebnom poglavlju.

5.1.1. Metode autentifikacije

Već kroz prijašnja poglavlja govoreno je kako postoje razne metode autentifikacije, a Firebase Authentication ih podržava sve uz iznimku za biometrijsku autentifikaciju. Uz to postoji i više pristupa samoj implementaciji autentifikacijskih rješenja kao što je korištenje FirebaseUI-a za implementaciju već izgrađenog sučelja ili jednostavno dodavanjem Firebase Authentication-a u projekt što će omogućiti implementaciju rješenja koja želimo.

Tako dakle uz standardnu autentifikaciju putem email adrese i lozinke na raspolaganju se još nalazi i podrška za implementaciju SSO tehnologija od kojih su najznačajnije Google, Facebook, Apple i GitHub, a za dodatnu sigurnost tu su i mogućnosti implementacije

autentifikacije putem unosa jednokratnog kôda koji se šalje na korisnikov mobitel. Uz to Firebase Authentication nudi i mogućnost definiranja vlastitih OAuth i OpenID poslužitelja.

5.1.2. Zašto koristiti Firebase Authentication

Glavna prednost ovog pristupa je lakoća njegove implementacije, neovisno radi li se o novoj ili postojećoj aplikaciji. Također na raspolaganju je pregršt izbora što je posebno značajno ako razvijamo aplikaciju za više platformi. Sigurnost je također nešto po čemu se ovaj pristup ističe. Kako je Firebase Authentication zasnovan na OAuth, OIDC i SAML protokolima, ovisno o funkcionalnosti koja se implementira, sigurnost podataka korisnika je zajamčena.

No glavni problem ovog pristupa je manjak kontrole koji programeri imaju. Ukoliko se programeri odluče za implementaciju Firebase Authenticationa postaju ovisni o Googleovim servisima te se ograničava njihova mogućnost za izmjenu samog procesa autentifikacije. Također problem može doći i u cijeni. Iako postoji besplatna opcija, ukoliko aplikacija pređe određeni broj korisnika ili želi koristiti određene funkcionalnosti onda sama platforma zahtjeva plaćanje.

Ukratko Firebase Authentication sjajna je platforma za implementaciju neophodnih funkcionalnosti u većini aplikacija bez potrebe za razvojem vlastitih kompliciranih rješenja. Korištenjem ovog pristupa programeri imaju više vremena i resursa te se mogu posvetiti ostalim funkcionalnostima aplikacije bez da korisnici gube na jednostavnosti korištenja iste.

5.2. SSO tehnologije

Iako je u prošlom poglavlju bilo govora o SSO tehnologijama i njihovoj implementaciji putem Firebase Authenticationa, to nije jedini način da se te tehnologije implementiraju u mobilne aplikacije. Isto kako Firebase Authentication pruža svoje razvojne okvire (eng. *software development kit – SDK*) za različite platforme, tako i mnoga poduzeća koja pružaju SSO usluge imaju vlastite SDK-ove za implementaciju njihovog pristupa autentifikacije i autorizacije u aplikacije.

Dvije najpopularnije tehnologije u ovom području su Google Sign-in i Facebook login. Google Sign-in metoda nedavno je ažurirana za mobilne uređaje te se tako sada odvija korištenjem Google Credential Managera, ali princip je isti. U pozadini se koristi OAuth 2.0 protokol koji jamči sigurnost za korisnike te im omogućuje da se u aplikaciju prijave koristeći svoj Google račun („Integrate Credential Manager with Sign in with Google“, bez dat.). Facebook login radi na sličan način. Doduše on ne koristi Credential Manager za implementaciju već vlastiti SDK putem kojeg komunicira sa aplikacijom koja se kreira putem

Facebook konzole za razvojne programere te se autentifikacija vrši tim putem („Android - Facebook Login - Documentation“, bez dat.). Naravno više o samoj implementaciji biti će govoreno u zasebnom poglavlju.

SSO tehnologije pružaju sjajno rješenje za implementaciju autentifikacije u aplikaciju putem korisničkih računa koje korisnik već posjeduje. Budući da se baziraju na OAuth protokolu povećana je i sigurnost autentifikacije, a programerima je implementacije ove funkcionalnosti olakšana.

No slično kao i kod Firebase Authentication-a važno je naglasiti kako je ovisnost o ovim poslužiteljima glavni problem. Međutim u ovom pristupu se Google i Facebook koriste samo kao dodatne metode autentifikacije u postojeću aplikaciju pa se programeri mogu odlučiti koje poslužitelje žele, a koje ne žele implementirati u svoju aplikaciju. Ovo pruža fleksibilnost u odabiru najboljeg pristupa za specifične potrebe aplikacije.

5.3. Auth0

Auth0 primjer je fleksibilnog centraliziranog gotovog rješenja za implementaciju autentifikacije i autorizacije u aplikaciju (Auth0, bez dat.-b). Prema načinu rada ova metoda slična je Firebase Authentication metodi po tome što nudi raznovrsne načine za implementaciju prijave u aplikaciju, a radi sa OAuth, OIDC i SAML protokolima. Tako je moguće implementirati klasičnu prijavu putem email adrese i lozinke, SSO pristup, te i raznovrsne pristupe za višerazinsku autentifikaciju poput biometrijske autentifikacije ili autentifikacije putem SMS kôda. Auth0 dostupan je za više platformi, a ističe se u svojoj uporabi u poduzećima.

Auth0 poznat je po svojoj fleksibilnosti i prilagodljivosti, što ga čini idealnim rješenjem za širok spektar korisnika, od malih poduzeća do velikih korporacija. Jedna od ključnih posebnosti Auth0-a je njegova sposobnost da podrži raznovrsne uređaje, uključujući senzore, aktuatore i druge uređaje koji spadaju pod Internet of Things (IoT) klasifikaciju. Ovo omogućava programerima implementaciju autentifikacije i autorizacije na raznovrsne platforme i za raznovrsne korisnike.

Kada je u pitanju integracija s poduzećima i B2B korisnicima Auth0 pruža raznovrsne metode integracije poput integracije s Active Directory, Google Workspace, Azure Active Directory Native i drugim uslugama (Auth0, bez dat.-c).

Uz to Auth0 iznimno je prilagodljiv potrebama korisnika. Tako se putem web aplikacije mogu prilagođavati SDK-ovi koje će programer na kraju implementirati u aplikaciju, a na raspolaganju je i nadzorna ploča (eng. *dashboard*) koja pruža vlasniku ili administratoru aplikacije detaljan uvid u korisničke profile, sigurnosne politike koje su trenutno primijenjene ili same performanse aplikacije.

Sve ovo razlozi su zašto izabrati Auth0 prilikom implementacije prijave u svoju aplikaciju. Velika razina fleksibilnosti te podrška za raznovrsne klijente i platforme kao i za raznovrsne veličine aplikacija razlozi su zašto se okrenuti ka ovoj tehnologiji. Međutim kao i Firebase Authentication ima svoje mane poput ovisnosti o platformi i cijeni kada se pređe određeni broj korisnika aplikacije ili kada je potrebno implementirati određene funkcionalnosti.

5.4. Koju tehnologiju izabrati

Nakon svih spomenutih tehnologija može se doći do zaključka da su svi pristupi slični i svi imaju određene prednosti i mane. U tablici dva sistematizirao sam spomenute tehnologije.

Tablica 2: Sistematizacija tehnologija za proces autentifikacije u aplikacijama

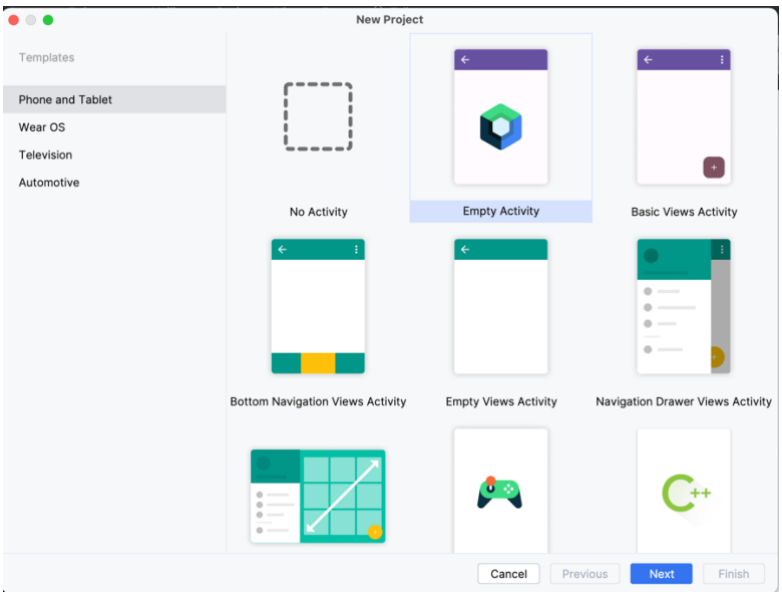
Mogućnost	**Firebase Authentication**	**SSO tehnologije**	**Auth0**
Podržane metode autentifikacije	Email/lozinka, SSO, Višerazinska autentifikacija	Samo SSO pristup	Email/lozinka, SSO, Višerazinska autentifikacija, Uporaba u poduzećima
Sigurnosni protokoli	OAuth 2.0, OIDC, SAML	OAuth 2.0, OIDC	OAuth 2.0, OIDC, SAML
Podržane platforme	Android, iOS, Web, Unity	Android, iOS, Web	Android, iOS, Web, IoT
Integracija	Detaljan SDK i jednostavna integracija u postojeće aplikacije ili kreiranje aplikacije od početka	SDK specifičan za tehnologiju koja se koristi, jednostavna integracija	Detaljan SDK i jednostavna integracija u postojeće aplikacije ili kreiranje aplikacije od početka uz pružanje gotovog predloška
Dodatne značajke	Integracija s ostalim Firebase uslugama	Specifično za tehnologiju koja se koristi	Prilagodljiva pravila, Nadzorna ploča za upravljanje korisnicima
Podrška za višerazinsku autentifikaciju	OTP, SMS	Ne	OTP, SMS, Biometrija
Skalabilnost	Visoka	Ovisno o implementaciji ali generalno visoka	Visoka
Kontrola nad korisničkim računima	Ograničena (Firebase upravlja računima)	Ograničena (računima upravlja ona tehnologija koja je implementirana)	Visoka razina kontrole
Primjena u postojećim aplikacijama	Jednostavna integracija u postojeće aplikacije	Jednostavna integracija u postojeće aplikacije	Jednostavna integracija u postojeće aplikacije
Trošak	Besplatno do određene razine korisnika i	Besplatno	Besplatno do određene razine korisnika i

funkcionalnosti, onda plaćeno		funkcionalnosti, onda plaćeno

Kao što je vidljivo iz tablice stvarno svaka tehnologija ima svoje određene prednosti i mane. Generalno gledano kada se radi o manjim aplikacijama koje želimo brzo razviti te se ne želimo zamarati vlastitim implementacijama za autentifikaciju i autorizaciju najbolja tehnologija je Firebase Authentication. Jednostavno ova tehnologija pruža jako širok spektar mogućnosti a jako je jednostavna za korištenje te ju podupire Google. S druge strane ako razvijamo aplikaciju većeg opsega, posebno ako mislimo raditi sa poduzećima i njihovim postojećim infrastrukturama najbolji izbor je Auth0. Zbog još većeg broja mogućnosti i prilagodljivosti za još veći broj platformi idealan je izbor ukoliko ne želimo sami razvijati vlastito rješenje. No to ne znači da je direktna implementacija SSO tehnologija u postojeće aplikacije nepoželjna jer ovaj pristup ostavlja veliku kontrolu u rukama programera koji razvijaju aplikaciju. No ova metoda je vjerojatno najsporija i ovisno o raznovrsnosti tehnologija koje planiramo integrirati može biti i najkompliciranija. Konačna odluka o načinu implementacije naravno ovisi o potrebama aplikacije i mogućnosti tima koji ju razvija. U idućem poglavlju biti će implementirana sva tri pristupa.

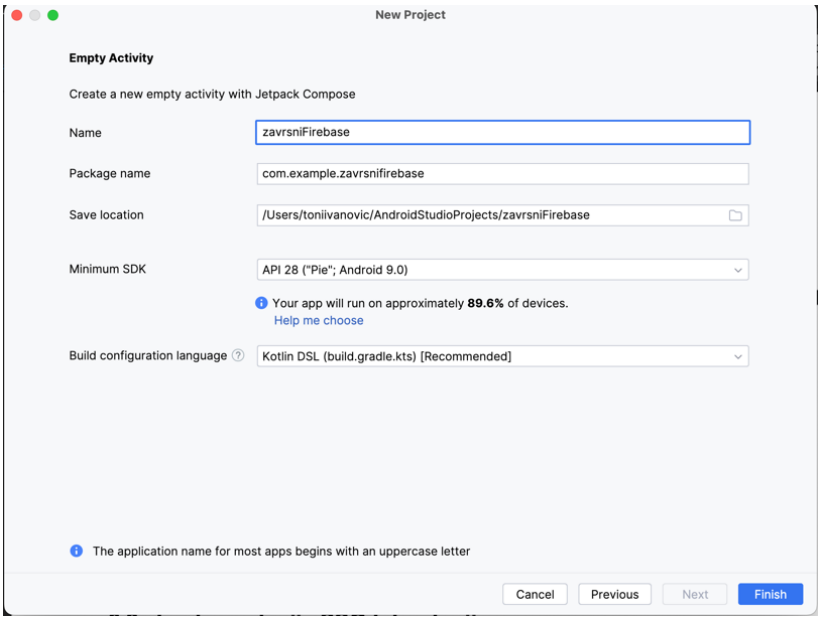
6. Implementacija tehnologija za autentifikaciju i autorizaciju u mobilnu aplikaciju

Do sada je bilo puno govora o teoriji kako o samim tehnologijama tako i o protokolima i procesima prijave i registracije. Prije nego se krene sa implementacijom specifičnih tehnologija potrebno je kreirati novi projekt u Android Studio razvojnom okruženju. Budući da je potrebno implementirati tri različite tehnologije, potrebno je kreirati i tri različita projekta. Kako je ovaj proces uvijek isti prikazati ću ga ovdje na primjeru za Firebase tehnologiju. Na slici četiri vidljiv je početni zaslon prilikom kreiranja novog projekta.



Slika 4: Početni zaslon prilikom kreiranja novog projekta u Android Studiju

Sljedeći korak je imenovanje projekta. Ovaj korak posebno je važan kada se radi sa aplikacijama trećih strana budući da nam o imenu projekta i imenu paketa ovisi korektna implementacija funkcionalnosti iz tih aplikacija. Na slici pet vidljiv je i taj zaslon.



Slika 5: Prikaz zaslona za imenovanje projekta u Android Studiju

U ovom dijelu izabiremo ime projekta što je ujedno i ime aplikacije. Ime paketa bitno je kada koristimo aplikacije trećih strana budući da se putem njega potvrđuje da se radi o ispravnoj aplikaciji, a standardan oblik je „com.organizacija.ime_aplikacije“. Sljedeći atribut koji biramo je minimalna verzija operacijskog sustava na kojem se aplikacija može pokrenuti što je u ovom slučaju Android 9. Zadnja opcija odnosi se na jezik u kojem će biti zapisana konfiguracija za pokretanje aplikacije koja je ostavljena na preporučenim postavkama. Nadalje potrebno je sačekati nekoliko sekundi kako bi se izvršio proces postavljanja projekta unutar kojeg Android Studio konfigurira specifikacije projekta i postavlja određene predloške u kôdu, a nakon toga projekt je spreman za rad (Griffiths & Griffiths, 2021, Poglavlje 1).

U nadolazećim potpoglavljima bit će implementirane sve tri spomenute tehnologije te će se na kraju poglavlja izvesti zaključak o implementaciji, odnosno svojevrstan osvrt na implementaciju spomenutih tehnologija. Sva implementacija rađena je po uzoru na standarde korištene u vrijeme pisanja završnog rada te prema dokumentaciji za razvojne programere koja se koristi kao literatura istoga. Kako bi se kôd mogao izvršavati potreban je razvojno okruženje Android Studio. Izvorni kôd svih poglavlja dostupan je na platformi GitHub na poveznici https://github.com/tivanovic21/foi-zavrsni.

6.1. Implementacija Firebase Authenticationa

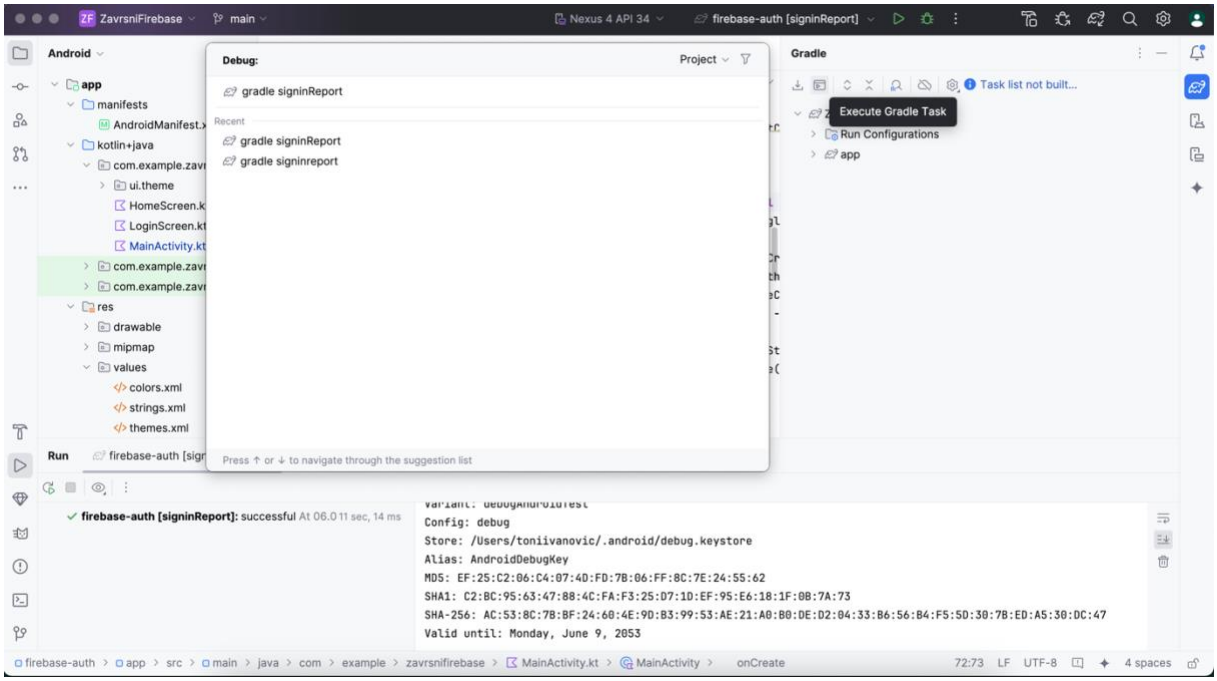
U ovom poglavlju biti će prikazani koraci za implementaciju Firebase Authenticationa u mobilnu aplikaciju. Fokus će biti na implementaciji registracije i prijave putem email adrese i lozinke te jedne SSO tehnologije, odnosno prijave koristeći Google račun. Izvorni kôd za ovo

potpoglavlje nalazi se u gore spomenutom repozitoriju u zasebnoj sekciji pod imenom „firebase-auth“.

6.1.1. Kreiranje novog projekta u Firebase konzoli

Kao što je već spomenuto prvi korak kod implementacije neke tehnologije u našu aplikaciju je kreiranje novog projekta, odnosno same aplikacije. Drugi korak u ovom slučaju je kreiranje novog projekta putem Firebase web aplikacije koja se nalazi na poveznici https://console.firebase.google.com. Ovdje je potrebno izabrati opciju „Get started with a Firebase project“ što će nam ponuditi opciju da unesemo ime projekta, a ja sam ovdje unijo ime „zavrsni-foi“. Dalje je samo potrebno slijediti korake koji se prikažu na zaslonu, a uglavnom se odnose na prihvaćanje uvjeta.

Kada je to napravljeno dočekat će nas zaslon za registraciju aplikacije. U njemu je bitno unijeti isto ime paketa koje smo koristili tijekom kreiranja našeg projekta u Android Studiju te SHA-1 sažetak aplikacije ako želimo implementirati određene funkcionalnosti poput Google Sign-ina. Također je moguće unijeti i ime aplikacije, ali je taj dio opcionalan. Ove informacije koristit će se za potvrdu da se Firebase projekt integrira u valjanu aplikaciju. Kako bi se dobio SHA-1 sažetak aplikacije potrebno je putem Android Studija pozicionirati se na „Gradle“ alat te preko nje komunicirati sa terminalom koristeći funkciju „signinReport“. Ovaj postupak vidljiv je na slici šest.

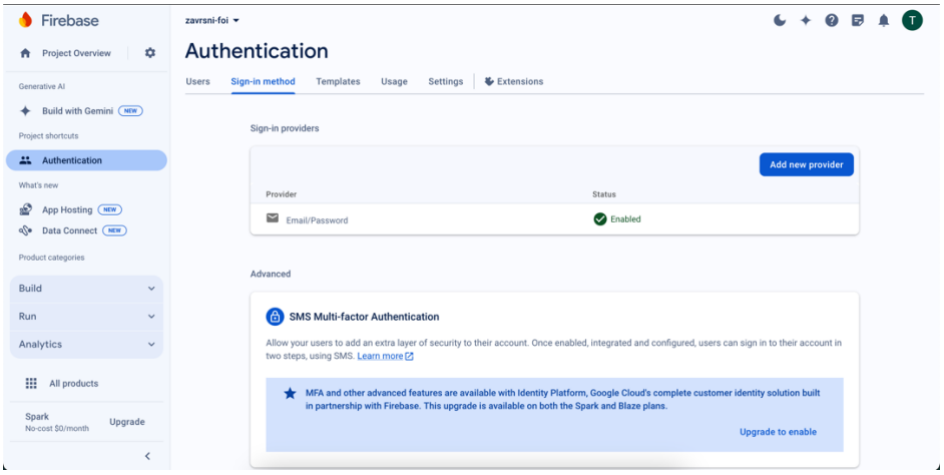


Slika 6: Prikaz procesa za dobivanje SHA-1 sažetka aplikacije putem Android Studija

Nakon registracije aplikacije generira se dokument sa konfiguracijskim podacima pod nazivom „google-services.json“ koji je potrebno postaviti u početni direktorij aplikacije. U tom dokumentu nalaze se važni podaci o klijentu i projektu koji će biti korišteni za autentifikaciju u daljnjim koracima. Zatim slijedi nekoliko koraka oko dodavanja referenci na taj dokument u našu aplikaciju. Nakon toga potrebno je izvršiti sinkronizaciju projekta (eng. sync) kako bi se preuzeli potrebni dokumenti. Tim korakom završava proces postavljanja projekta.

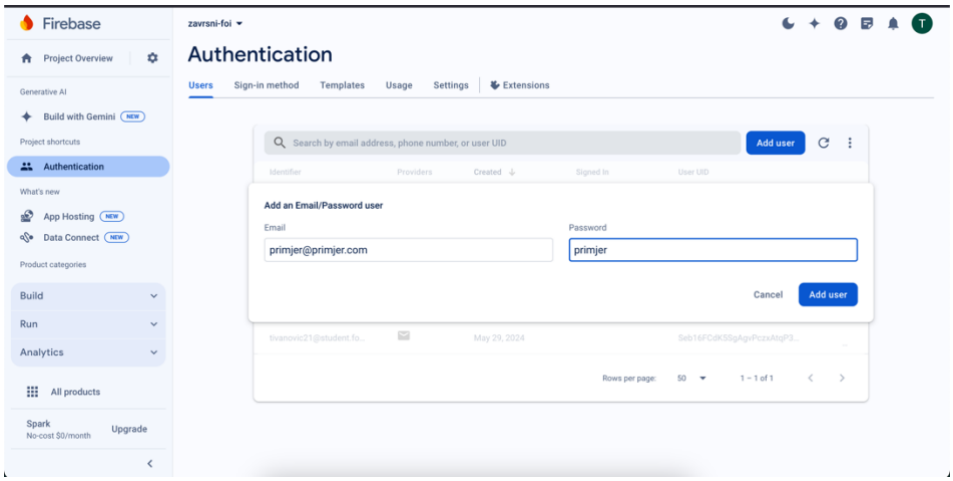
6.1.2. Implementacija početne metode za prijavu i registraciju

Kako bi omogućili prijavu putem Firebase Authenticationa u aplikaciju potrebno je postaviti opcije za autentifikaciju putem Firebase konzole. Na slici sedam vidljivo je kako konzola izgleda kada je odabran jedan od pružatelja ove usluge.



Slika 7: Prikaz Firebase konzole sa odabranom metodom za autentifikaciju

Kao prvu metodu izabrao sam autentifikaciju putem email adrese i lozinke. Jednom kada je metoda odabrana putem konzole u sekciji „Users“ moguće je dodati novog korisnika. Proces dodavanja prikazan je na slici osam.



Slika 8: Prikaz Firebase konzole za dodavanje novog korisnika

Kada se doda novi korisnik njegova lozinka sama se sažima i sigurno pohranjuje u bazu podataka na Firebase-u te se korisnik može prijaviti u našu aplikaciju sa podacima koji su korišteni prilikom dodavanja. Putem konzole također možemo vidjeti točne parametre koji su korišteni za sažimanje lozinke. Za potrebe ovog završnog rada ja sam dodao korisnika sa email adresom tivanovic21@student.foi.hr i lozinkom „firebase1“.

Sada kada imamo korisnika u Firebase konzoli te smo već podesili ovisnosti u aplikaciji kada je riječ o Firebaseu potrebno je samo napisati kôd kako bi mogli provjeriti autentičnost korisnika prilikom ulaska u našu aplikaciju.

Budući da smo kreirali projekt koristeći „Empty Activity“ predložak Android studio generirao je određene direktorije i dokumente na osnovu kojih možemo nastaviti našu implementaciju (Griffiths & Griffiths, 2021, Poglavlje 1). Najbitniji predložak za nastavak implementacije Firebase Authentication-a je trenutno MainActivity klasa. U tu klasu treba dodati varijablu koja će se odnositi na autentifikaciju putem Firebase Authentication-a te kako se radi o Compose pristupu dizajna aplikacije potrebno je imati funkciju unutar koje će se nalaziti dizajn i funkcionalnost zaslona za prijavu. Kako izgleda ta klasa vidljivo je u prvom odjeljku programskog kôda.

```
1. class MainActivity : ComponentActivity() {
2.
3.   private lateinit var auth: FirebaseAuth
4.
5.   override fun onCreate(savedInstanceState: Bundle?) {
6.       super.onCreate(savedInstanceState)
7.       FirebaseApp.initializeApp(this)
8.       auth = Firebase.auth
9.
10.        setContent {
11.            ZavrsniFirebaseTheme {
```

```
                Scaffold(modifier = Modifier.fillMaxSize()) {
                innerPadding ->
                    AuthScreen(
                        auth = auth,
                        modifier = Modifier.padding(innerPadding)
                    )
                }
            }
        }
    }
}
```

Programski kôd 1: MainActivity dodavanje zaslona za autentifikaciju

Ova klasa razlikuje se od standardne klase koja se dobije generiranjem projekta samo po varijabli „auth“ koja služi kako bi se komuniciralo sa Firebase Authentication klasom i „AuthScreen“ metodi koja je zaslužna za prikaz zaslona za prijavu. Implementacija ove metode vidljiva je u drugom programskom odlomku.

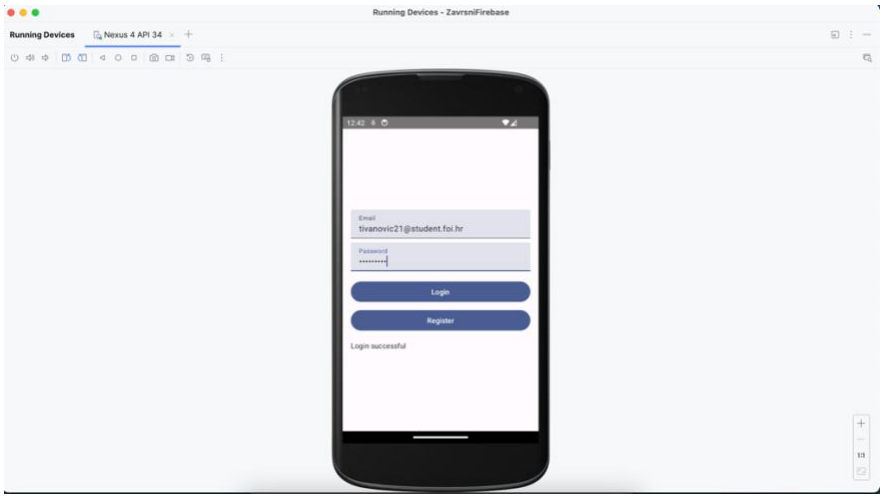
```
@Composable
fun AuthScreen(auth: FirebaseAuth, modifier: Modifier = Modifier) {
    var email by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var message by remember { mutableStateOf("") }

    Column(
        modifier = modifier.fillMaxSize().padding(16.dp),
        verticalArrangement = Arrangement.Center
    ) {
        TextField(
            value = email,
            onValueChange = { email = it },
            label = { Text("Email") },
            modifier = Modifier.fillMaxWidth()
        )
        Spacer(modifier = Modifier.height(8.dp))
        TextField(
            value = password,
            onValueChange = { password = it },
            label = { Text("Password") },
            modifier = Modifier.fillMaxWidth(),
            visualTransformation = PasswordVisualTransformation()
        )
        Spacer(modifier = Modifier.height(16.dp))
        Button(
           onClick = {
                auth.signInWithEmailAndPassword(email, password)
                 .addOnCompleteListener { task ->
                  message = if (task.isSuccessful) {
                    "Login successful"
                   } else {"Login failed:
                     ${task.exception?.message}"
                   }
                 }
            },
```

```
35.                 modifier = Modifier.fillMaxWidth()
36.             ) {
37.                 Text("Login")
38.             }
39.             Spacer(modifier = Modifier.height(8.dp))
40.             Button(
41.                 onClick = {
42.                     auth.createUserWithEmailAndPassword(email,
                        password)
43.                       .addOnCompleteListener { task ->
44.                           message = if (task.isSuccessful) {
45.                               "Registration successful"
46.                           } else {
47.                               "Registration failed:
                                  ${task.exception?.message}"
48.                           }
49.                       }
50.                 },
51.                 modifier = Modifier.fillMaxWidth()
52.             ) {
53.                 Text("Register")
54.             }
55.             Spacer(modifier = Modifier.height(16.dp))
56.             Text(text = message,
                     style = MaterialTheme.typography.bodyMedium)
57.         }
58.     }
```

Programski kôd 2: AuthScreen metoda za imlpementaciju prijave i registracije

U sklopu ove metode napisan je osnovni izgled korisničkog sučelja te su dodane funkcionalnosti prijave i registracije u aplikaciju. Na liniji 30 vidljiv je poziv metoda signInWithEmailAndPassword. Ta metoda komunicira direktno sa Firebase-om te provjerava postoji li korisnik u bazi podataka sa podacima koji odgovaraju onima koji su uneseni. Dakle ukoliko korisnik unese podatke koje sam gore naveo dobit će poruku „Login successful“ te na osnovu te provjere korisniku se može dopustiti pristup aplikaciji, a ako ne onda će dobiti poruku o neuspješnoj prijavi sa popratnim tekstom o vrsti greške koja se dogodila. Ova funkcija pokriva razne greške od krivo unesenih podataka do neispravno formatirane email adrese. Također na liniji 46 vidljiv je poziv metode createUserWithEmailAndPassword, a ta metoda uzima podatke koje korisnik unese i pokušava kreirati novog korisnika. Ako se korisnik uspješno kreira ispiše se potvrdna poruka „Registration successful“, a ukoliko dođe do greške kao i kod prijave ispisuje se poruka o neuspješnoj registraciji sa popratnim tekstom o specifičnoj grešci koja se dogodila. Kada se korisnik uspješno registrira njegovi podaci vidljivi su u Firebase konzoli te je nad njim moguće raditi sve iste akcije kao i nad korisnikom koji je direktno kreiran putem konzole. Kako izgleda uspješna prijava u aplikaciju vidljivo je na slici devet.

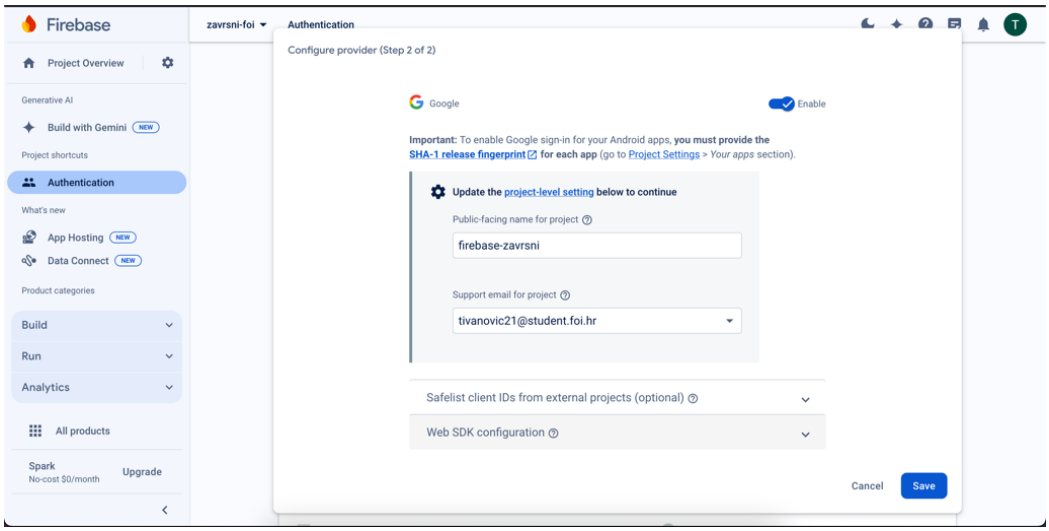


Slika 9: Početni zaslon nakon uspješne prijave

Ovim putem je dakle u manje od 100 linija kôda uspješno implementirana cijela logika za prijavu i registraciju u aplikaciju, bez ikakve potrebe za vlastitim serverom ili back-end kôdom.

6.1.3. Implementacija prijave koristeći Google račun putem Firebase Authentication tehnologije

Kao što je prikazano u prethodnom potpoglavlju implementacija autentifikacije putem Firebase Authentication tehnologije ne zahtjeva previše programiranja. Isti je slučaj i kada se radi o implementaciji SSO poslužitelja. Kako je bio slučaj i kod implementacije autentifikacije putem email adrese i lozinke tako i ovdje, potrebno je preko Firebase konzole izabrati SSO poslužitelja te omogućiti autentifikaciju njegovim putem. Ovaj korak prikazan je na slici deset.



Slika 10: Prikaz Firebase konzole za dodavanje Google Sign-in mogućnosti

Nakon što spremimo promjene potrebno je preuzeti novi dokument sa konfiguracijskim podacima i zamijeniti stari. On osigurava kako sada aplikacija ima potrebne informacije o radu sa Google-om kao autorizacijskim poslužiteljem. („Authenticate with Google on Android | Firebase Authentication“, bez dat.). Sada je važno za naglasiti kako račun koji sam kreirao ranije putem Firebase konzole više neće moći biti autentificiran tim putem, već isključivo putem Google Sign-in metode.

Implementacija prijave putem Google računa omogućava nam prikaz podataka o korisniku poput njegove slike, imena i slično. Kako bi prikazao te mogućnosti ažurirati ću osnovnu strukturu projekta te će sada uz klasu MainActivity postojati još dvije klase, klasa LoginScreen i klasa HomeScreen. Klasa LoginScreen zaslužna je za rad s različitim metodama prijave i registracije u aplikaciju te se ona otvara prva, a na klasi HomeScreen prikazivati će se podaci koji su nam dostupni o korisniku.

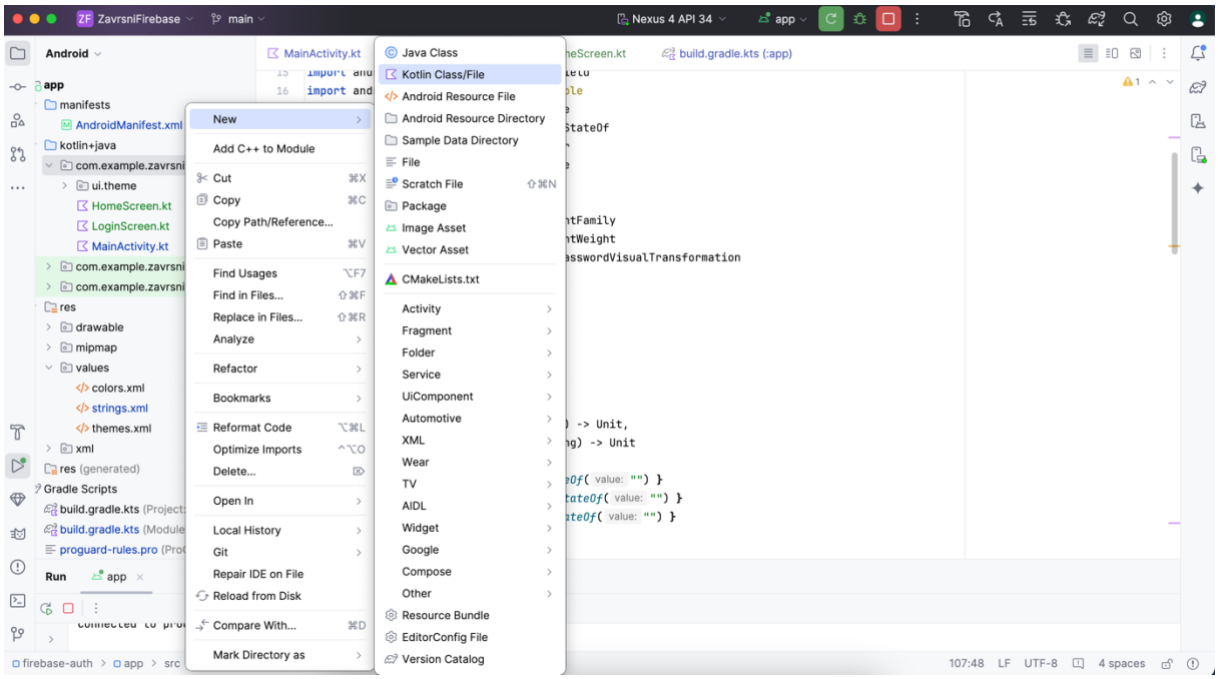
Prije početka implementacije važno je prilagoditi build.gradle.kts dokument kako bi se dodale nove ovisnosti. U trećem isječku kôda prikazane su nove ovisnosti, a uz njih se nalaze i komentari kako bi se naznačilo zašto su dodane.

```
1. implementation("androidx.navigation:navigation-compose:2.7.7") //
   biblioteka za navigaciju
2. implementation("io.coil-kt:coil-compose:2.6.0") // biblioteka za
   učitavanje slika
3.
4. // credential manager ovisnosti
5. implementation("androidx.credentials:credentials:1.2.2")
6. implementation("androidx.credentials:credentials-play-services-
   auth:1.2.2")
7.
   implementation("com.google.android.libraries.identity.googleid:google
   id:1.1.0")
```

Programski kôd 3: Ovisnosti za Credential Manager i Google Sign-in mogućnost autentifikacije

Nakon dodavanja ovih ovisnosti potrebno je ponovno izvršiti sinkronizaciju projekta. Radi estetskih razloga i lakše navigacije projektom dodane su biblioteke za navigaciju i učitavanje slika, a credential manager ovisnosti značajne su za samu implementaciju Credential Manager-a i prijave putem Google računa.

Prije nego što išta promijenimo u MainActivity klasi potrebno je kreirati LoginScreen i HomeScreen klase. Na slici 11 prikazano je kako doći do izbornika za kreiranja nove klase.



Slika 11: Prikaz procesa za dodavanje nove klase u Android Studiju

Klase je potrebno nazvati LoginScreen i HomeScreen. Prvo ću krenuti sa implementacijom LoginScreen klase jer je ona upravo ta koja se i treba otvoriti prilikom pokretanja aplikacije. Dosta kôda preuzeti ću iz početne AuthScreen metode budući da su zadaće slične.

Dakle prvo je potrebno definirati Composable metodu sa nazivom LoginScreen koja će primati modifier atribut i tri druge metode. Te tri metode biti će odgovorne za prijavu putem email adrese i lozinke, registraciju novog korisničkog računa putem email adrese i lozinke te za prijavu putem Google računa. U odsječku programskog kôda četiri vidljivo je kako metoda treba izgledati u svom osnovnom obliku.

```
1. @Composable
2. fun LoginScreen(
3.     modifier: Modifier = Modifier,
4.     onGoogleSignInClick: () -> Unit,
5.     onEmailSignInClick: (String, String) -> Unit,
6.     onEmailRegisterClick: (String, String) -> Unit
7. ) {}
```

Programski kôd 4: Osnovna implementacija LoginScreen metode

Unutar ove klase piše se samo dizajn sučelja, odnosno kako se informacije trebaju prikazivati na zaslonu. Zato su nam bitne metode koje se prosljeđuju kao atributi jer će se njih pozivati u LoginScreen klasi, ali će njihove funkcionalnosti biti definirane u MainActivity klasi. Primjer kako izgleda takav poziv metode nalazi se u petom programskom odsječku.

```
var email by remember { mutableStateOf("") }
var password by remember { mutableStateOf("") }

Button(
    onClick = { onEmailSignInClick(email, password) },
    modifier = Modifier.fillMaxWidth()
) {
   Text(
        text = "Login with Email",
         fontFamily = FontFamily.Monospace,
         fontWeight = FontWeight.SemiBold,
         fontSize = 20.sp
    )
 }
```

Programski kôd 5: Poziv metode za prijavu putem email adrese i lozinke u LoginScreen metodi

Na isti princip funkcionira i implementacija HomeScreen klase, ali u nju je potrebno proslijediti i korisnika koji je trenutno prijavljen kako bi se njegovi podaci mogli koristiti za prikaz. Prijavljeni korisnik objekt je klase FirebaseUser te sadrži podatke o korisniku, poput njegove slike, imena, email adrese i sličnih podataka te će upravo preko njega biti pristupano podacima korisnika. Kako izgleda osnovna implementacija HomeScreen metode zajedno sa jednim primjerom programske logike za dohvaćanje podataka korisnika vidljivo je u šestom programskom odsječku.

```
@Composable
fun HomeScreen(modifier: Modifier = Modifier,
               currentUser: FirebaseUser?,
               onSignOutClick: () -> Unit) {
    Surface(
        modifier = modifier.fillMaxSize()
    ){
      Column(
          modifier = modifier.fillMaxSize()
                .padding(horizontal = 16.dp),
          horizontalAlignment = Alignment.CenterHorizontally,
          verticalArrangement = Arrangement.Center
      ) {
          currentUser?.let { user ->
            user.photoUrl?.let {
                AsyncImage(
                    modifier = Modifier.size(140.dp)
                        .clip(RoundedCornerShape(4.dp)),
                    model =
                     ImageRequest.Builder(LocalContext.current)
                        .data(it).crossfade(true).build(),
                    contentDescription = "profile picture",
                    contentScale = ContentScale.Crop
                )
                Spacer(modifier = Modifier.size(16.dp))
            }
          }
      }
    }
```

```
29.  }
```

Programski kôd 6: Implementacija HomeScreen klase uz primjer dohvaćanja slike profila korisnika

Iz programskog kôda vidimo kako klasa kod poziva prima atribute modifier, currentUser i funkciju za odjavu. U ovom kontekstu currentUser može biti null vrijednosti, a time se osigurava da ako pozovemo ovu klasu bez valjane instance trenutnog korisnika ne dođe do rušenja same aplikacije. Na liniji 16 ovog programskog kôda provodi se prva provjera, odnosno provjerava se postoji li korisnik zaista. Ukoliko korisnik postoji preko njega provjeravaju se podaci koji su o njemu dostupni. Tako je primjerice photoUrl podatak koji također može biti null, što će se kasnije i prikazati kada se korisnik prijavi putem email adrese i lozinke no ukoliko slika profila nije null provodi se logika za prikaz te slike. U ovom slučaju koristi se asinkrona composable funkcija iz biblioteke Coil koju smo ranije uključili kao jednu od ovisnosti. Na isti ovaj način mogu se prikazati i ostale vrijednosti poput email adrese, imena i slično. Kada smo zadovoljni sa ovima dvama klasama možemo se vratiti na MainActivity klasu kako bi implementirali logiku za prijavu i registraciju korisnika.

Klasa MainActivity sada se značajno razlikuje od prvobitne implementacije pa je cijela vidljiva u sedmom programskom isječku.

```
1. enum class Screen{
2.     Login,
3.     Home
4. }
5. class MainActivity : ComponentActivity() {
6.
7.     private lateinit var auth: FirebaseAuth
8.     private lateinit var WEB_CLIENT_ID: String
9.
10.     override fun onCreate(savedInstanceState: Bundle?) {
11.         super.onCreate(savedInstanceState)
12.         FirebaseApp.initializeApp(this)
13.         auth = Firebase.auth
14.         WEB_CLIENT_ID = getString(R.string.default_web_client_id)
15.         enableEdgeToEdge()
16.         setContent {
17.             ZavrsniFirebaseTheme {
18.                 val navController: NavHostController =
                    rememberNavController()
19.                 val context = LocalContext.current
20.                 val scope = rememberCoroutineScope()
21.                 val credentialManager: CredentialManager =
                    CredentialManager.create(context)
22.
23.                 val startDestination = if (auth.currentUser ==
                    null) { Screen.Login.name } else Screen.Home.name
24.
25.                 NavHost(navController = navController,
                    startDestination = startDestination) {
26.                     composable(Screen.Login.name) {
27.                         LoginScreen(
```

```
28.                         onGoogleSignInClick = {
29.                             val googleIdOption =
                                GetGoogleIdOption.Builder()
30.                              .setFilterByAuthorizedAccounts(false)
31.                               .setServerClientId(WEB_CLIENT_ID)
32.                                .build()
33.
34.                              val request =
                                 GetCredentialRequest.Builder()
35.                               .addCredentialOption(googleIdOption)
36.                                .build()
37.
38.                               scope.launch {
39.                                 try {
40.                                   val result =
                                      credentialManager.getCredential(
41.                                              context = context,
42.                                              request = request
43.                                    )
44.                                    val credential =
                                       result.credential
45.                                    val googleIdTokenCredential =
                                       GoogleIdTokenCredential
46.                                     .createFrom(credential.data)
47.                                    val googleIdToken =
                                       googleIdTokenCredential
                                       .idToken
48.                                     val firebaseCredential =
                                        GoogleAuthProvider
                                        .getCredential
                                        (googleIdToken, null)
49.
                                    auth.signInWithCredential
                                    (firebaseCredential)
50.                                  .addOnCompleteListener {
                                     task ->
51.                                    if (task.isSuccessful) {
52.                                    navController.popBackStack()
53.                                    navController.navigate
                                        (Screen.Home.name)
54.                                    }
55.                                  }
56.                                } catch (e: Exception) {
57.                                    Toast.makeText(context,
                                       "Error: ${e.message}",
                                       Toast.LENGTH_SHORT).show()
58.                                }
59.                             }
60.                         },
61.                         onEmailSignInClick = { email, password
                            ->
62.                         auth.signInWithEmailAndPassword(email,
                            password)
63.                          .addOnCompleteListener { task ->
64.                            if (task.isSuccessful) {
65.                                navController.popBackStack()
66.                                navController.navigate
                                   (Screen.Home.name)
67.                             } else {
68.                               Toast.makeText(context, "Login
                                  failed:
```

```
                                        ${task.exception?.message}",
                                        Toast.LENGTH_SHORT).show()
69.                                     }
70.                                   }
71.                                 },
72.                                 onEmailRegisterClick = { email,
                                    password ->
73.                                  auth.createUserWithEmailAndPassword
                                     (email, password)
74.                                   .addOnCompleteListener { task ->
75.                                     if (task.isSuccessful) {
76.                                       navController.popBackStack()
77.                                       navController.navigate
                                          (Screen.Home.name)
78.                                     } else {
79.                                        Toast.makeText(context,
                                          "Registration failed:
                                          ${task.exception?.message}",
                                          Toast.LENGTH_SHORT).show()
80.                                     }
81.                                   }
82.                                 }
83.                             )
84.                         }
85.                         composable(Screen.Home.name) {
86.                            HomeScreen(
87.                                currentUser = auth.currentUser,
88.                                onSignOutClick = {
89.                                   auth.signOut()
90.                                   scope.launch {
91.                                     credentialManager
                                        .clearCredentialState(
92.                                        ClearCredentialStateRequest()
93.                                     )
94.                                   }
95.                                   navController.popBackStack()
96.                                   navController
                                      .navigate(Screen.Login.name)
97.                                }
98.                            )
99.                         }
100.                    }
101.                }
102.            }
103.        }
104. }
```

Programski kôd 7: Prikaz MainActivity klase koja implementira logiku za prijavu i registraciju putem email adrese i lozinke te prijavu putem Google računa

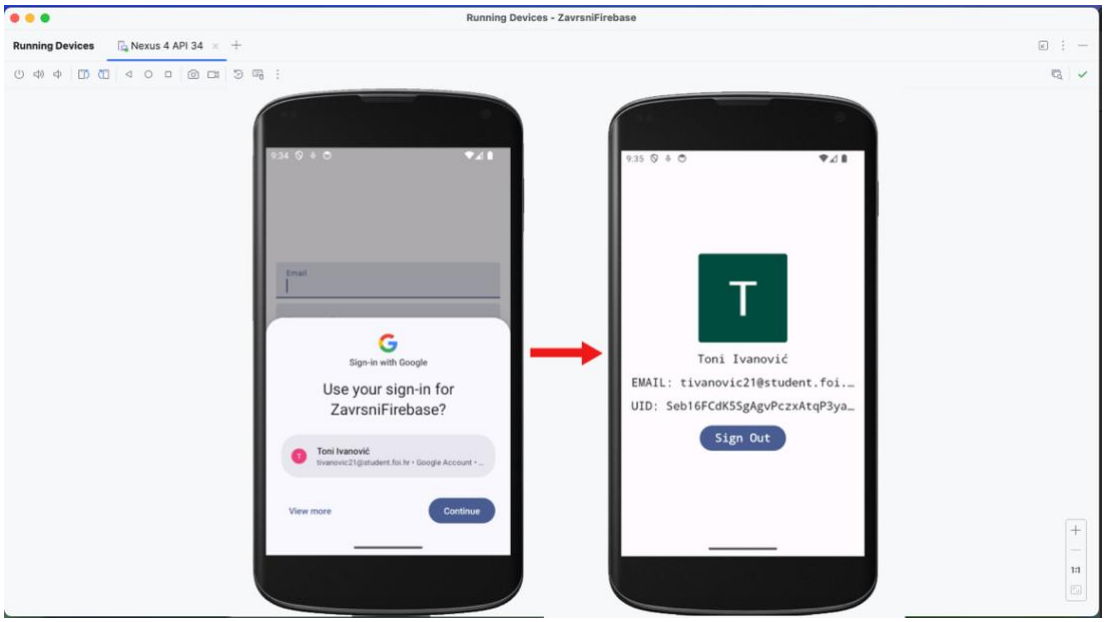
Prvo prije nego krenemo u samu MainActivity klasu treba naznačiti novu enumeraciju koja je dodana. Ta enumeracija služi za lakše identificiranje zaslona koje smo implementirali, u ovom slučaju Login i Home zaslona. Na ovaj način lakše je pratiti samu implementaciju te ju kasnije eventualno mijenjati ovisno o potrebi. Nadalje slijedi prva promjena u MainActivity klasi, odnosno WEB_CLIENT_ID varijabla. Ova varijabla odgovorna je kako bi se prilikom procesa autorizacije preko Google računa proslijedio valjani id klijenta te kako bi autorizacija mogla biti

valjano izvršena. Kako se id vrijednost ne bi spremala direktno u ovo klasu ona se čita iz dokumenta koji se automatski generira ukoliko uključimo valjani konfiguracijski dokument u našu aplikaciju. Zbog ovoga je bilo važno zamijeniti stari konfiguracijski dokument sa novim, jer ovaj sadrži podatke za autorizaciju s Google poslužiteljem. Sljedeća velika promjena je varijabla credentialManager koja predstavlja objekt istoimene klase. Ona služi kako bi se implementirala funkcionalnost Credential Manager-a putem kojeg se korisnici mogu prijaviti u aplikaciju koristeći svoje Google račune. Prva metoda koja koristi ovu varijablu je onGoogleSignInClick metoda. Ta metoda prije svega postavlja vjerodajnicu prema id-u klijenta koji čita iz prije spomenute varijable, a zatim se ta vjerodajnica koristi u zahtjevu za rad s Credential Managerom. Tu se nalazi logika za vađenje Id tokena kako bi se isti koristio za autentifikaciju s Firebase-om, a ako ta autentifikacija uspije korisnika se prebacuje na početni, Home, zaslon. Implementacijska logika za prijavu i registraciju uzeta je iz prijašnjeg primjera, a dorađena je tako da također korisnika prebacuje na početni zaslon ukoliko autentifikacija uspije. Druga metoda koja koristi credentialManager varijablu jest metoda za odjavu koja briše sesiju kako bi se naznačilo da je korisnik odjavljen, a uz to koristi se signOut metoda iz FirebaseAuth klase koja čisti podatke o korisniku iz cache pohrane. Bitno je pozvati obje metode kako bi i Credential Manager i FirebaseAuth bili obavješteni o odjavi korisnika.

Ovisno o programeru Credential Manager dopušta spremanje i drugih vjerodajnica te bi tako bilo moguće spremiti email adresu i lozinku korištenu za prijavu i onda korisnik ne bi morao stalno upisivati te podatke iznova već bi mogao koristiti podatke iz Credential Managera no to nije implementirano.

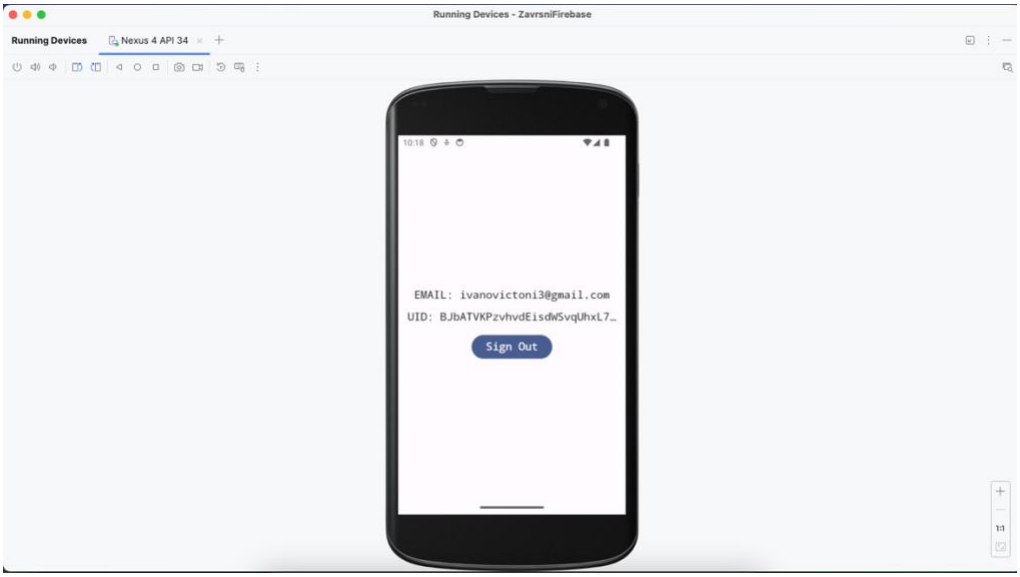
Tako u ovom slučaju Credential Manager je zaslužan i za komunikaciju sa Google računom, a potencijalno može služiti i kao svojevrsna baza podataka za sve vjerodajnice koje korisnik ima te koje koristi između različitih aplikacija.

Na slici 12 vidljiv je proces prijave koristeći Google račun i Credential Manager te početni zaslon nakon uspješne prijave.



Slika 12: Prikaz procesa prijave u aplikaciju koristeći Google račun putem Firebase-a

Kao što je prikazano na slici, a i ranije spominjano u implementaciji moguće je vidjeti sliku korisnika, ime i prezime, njegov email i uid token, a sve to samo su neki od svih dostupnih atributa. Za potrebe ovog primjera također sam registrirao novog korisnika sa email adresom ivanovictoni3@gmail.com i test12 lozinkom te je na slici 13 vidljiv početni zaslon nakon prijave u aplikaciju sa ovim računom.



Slika 13: Prikaz početnog zaslona nakon prijave u aplikaciju koristeći email adresu i lozinku putem Firebase-a

Kako je i na slici vidljivo sada imamo puno manje informacija o korisniku. Odnosno vidljiv je njegov uid token i email, ali nemamo podatke o njegovom imenu, prezimenu, slici ili

bilo čemu drugome. Odnosno sve što zapravo možemo znati o ovome korisniku je to što vidimo na slici.

Iz ovog poglavlja vidljivo je da je cijela implementacija opet zapravo dosta jednostavna, odnosno sva logika oko autentifikacije i autorizacije odrađena je putem Firebase Authentication tehnologije, a isto kao i u prvom primjeru podaci o korisniku vidljivi su na Firebase konzoli. Kako se implementacija ove tehnologije može usporediti sa implementacijama drugih tehnologija biti će govoreno u nadolazećem poglavlju.

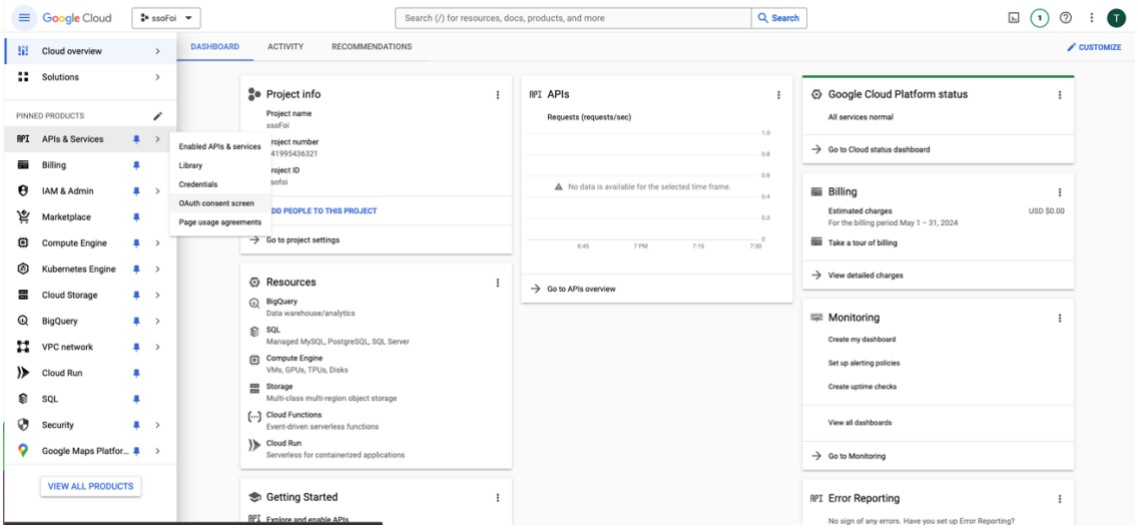
6.2. Implementacija SSO tehnologija

U prethodnom poglavlju implementirao sam Google Sign-in koristeći Firebase Authentication okvir. Međutim iako danas Firebase postaje standardizirano rješenje za rad mobilnih aplikacija, SSO tehnologije moguće je implementirati i u aplikaciju koja ne koristi Firebase. Kako bi se zadržao u okviru Kotlin programskog jezika, u ovom poglavlju implementirati će se Google Sign-in i Facebook login tehnologije, kako bi se prikazale sličnosti i razlike u implementaciji.

Kao i u prethodnom poglavlju kreiran je novi projekt sa nazivom „zavrsniSSO“, a izvorni kôd nalazi se u direktoriju sa nazivom „sso-auth“ na već spomenutom repozitoriju na GitHub-u.

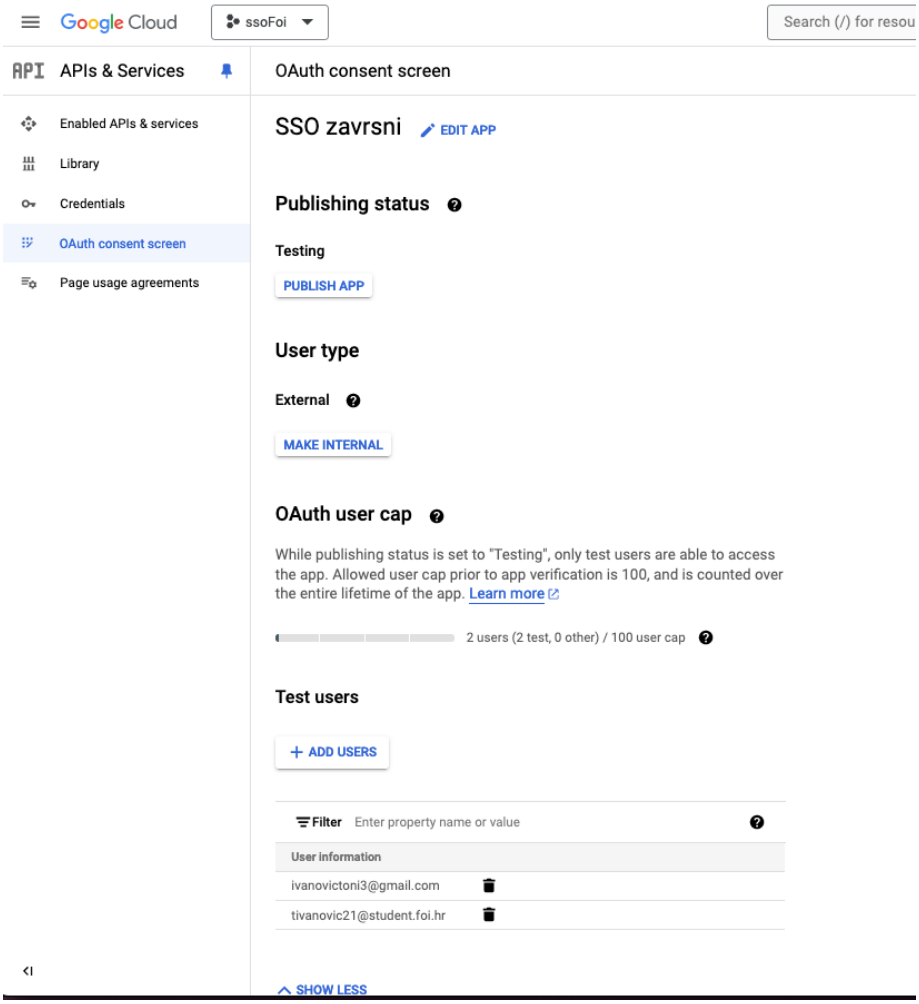
6.2.1. Implementacija prijave putem Google računa

Kao i prošli projekt i ovaj je podijeljen na MainActivity, LoginScreen i HomeScreen klase. Tako da je prvi korak oko dodavanja tih klasa i implementacije ovisnosti isti kao i prije. I u ovom primjeru koristiti će se Credential Manager jer je on danas preporučeni način za implementaciju Google Sign-in metode. Ovaj puta kako bi kreirali OAuth aplikaciju s kojom ćemo komunicirati kako bi prijava putem Google računa bila moguća potrebno je otići na Google Cloud konzolu i kreirati novi projekt. Konzola se nalazi na poveznici https://console.cloud.google.com/ i za kreaciju novog projekta potrebno je stisnuti gumb „New project“ te mu dati naziv. Ja sam projekt nazvao „ssoFoi“ te jednom kada se projekt kreira dočeka nas velika konzola sa pregršt informacija. Ono što je za sada bitno je pozicionirati se na „OAuth Consent Screen“ dio konzole kako bi mogli krenuti sa izradom OAuth aplikacije. Kako se pozicionirati na taj zaslon prikazano je na slici 14.



Slika 14: Prikaz putanje do OAuth Consent Screen dijela Google konzole

Kada se pozicioniramo na taj dio aplikacije potrebno je odabrati vrstu korisnika kojoj želimo dozvoliti rad s našom aplikacijom. Ja sam izabrao „External“ jer želim dozvoliti mogućnost korištenja aplikacije svim korisnicima sa Google računom, ne samo onima koji su unutar FOI organizacije. Nakon toga potrebno je dati ime aplikaciji kako bi korisnici znali koja aplikacija ih traži dopuštenje za pristup podacima te kontakt email adresu kako bi se korisnici imali kome obratiti u slučaju problema. Aplikaciju sam nazvao „SSO zavrsni“ te sam dao email adresu povezanu sa svojim računom. Nakon toga moguće je definirati opsege unutar aplikacije. Kako bi ovaj projekt bio što jednostavniji za korištenje ovdje nisam dodavao posebna ograničenja. Zatim je potrebno dodati korisnike koji će testirati aplikaciju. Ovdje sam dodao Google račune koji su korišteni i u ranijem projektu. Kako sve izgleda kada je aplikacija u potpunosti kreirana vidljivo je na slici 15.



Slika 15: Prikaz postavljene OAuth aplikacije putem Google Cloud konzole

Sada kada imamo OAuth aplikaciju potrebno je prebaciti se na sekciju „Credentials“ te kreirati nove vjerodajnice za OAuth Client ID aplikaciju. Iako bi se možda činilo intuitivno izabrati Android tip aplikacije, kako bi ova funkcionalnost radila potrebno je izabrati Web application tip aplikacije. Potrebno je unijeti ime aplikacije te kreirati aplikaciju. Nakon toga generira se JSON dokument koji sadrži podatke potrebne za implementaciju funkcionalnosti, ali najbitniji podatak nama je id klijenta. Taj id klijenta koristit će se kako bi se proveo proces autentifikacije putem Google računa. Kako bi se sigurno pohranio preporuča ga se izdvojiti u neki dokument pa sam ja tako izdvojio ga u strings.xml dokument sa nazivom „default_web_client_id“. U nadolazećem isječku kôda prikazati ću kako se isti dohvaća.

Kao što sam prije naglasio i ovaj put imamo LoginScreen i HomeScreen klase koje su po svojoj implementaciji slične klasama iz prošlog primjera, ali je ovdje potrebno napraviti i „data“ tip klase putem kojeg ćemo definirati objekt za prijavljenog korisnika. To se radi na isti način kao i kada se kreira LoginScreen i HomeScreen klasa, samo se mora paziti na to da se odabere „data“ tip. Ako se podsjetimo, kada smo koristili Firebase imali smo FirebaseUser

objekt preko kojeg smo iščitavali podatke o prijavljenom korisniku. Kako sada to nemamo potrebno je kreirati novu klasu, a implementacija ove klase prikazana je u osmom isječku kôda.

```
1. data class CurrentGoogleUser(
2.     val googleIdToken: String?,
3.     val displayName: String?,
4.     val photoUri: Uri?
5. )
```

Programski kôd 8: Implementacija data klase za prijavljenog korisnika putem Google računa

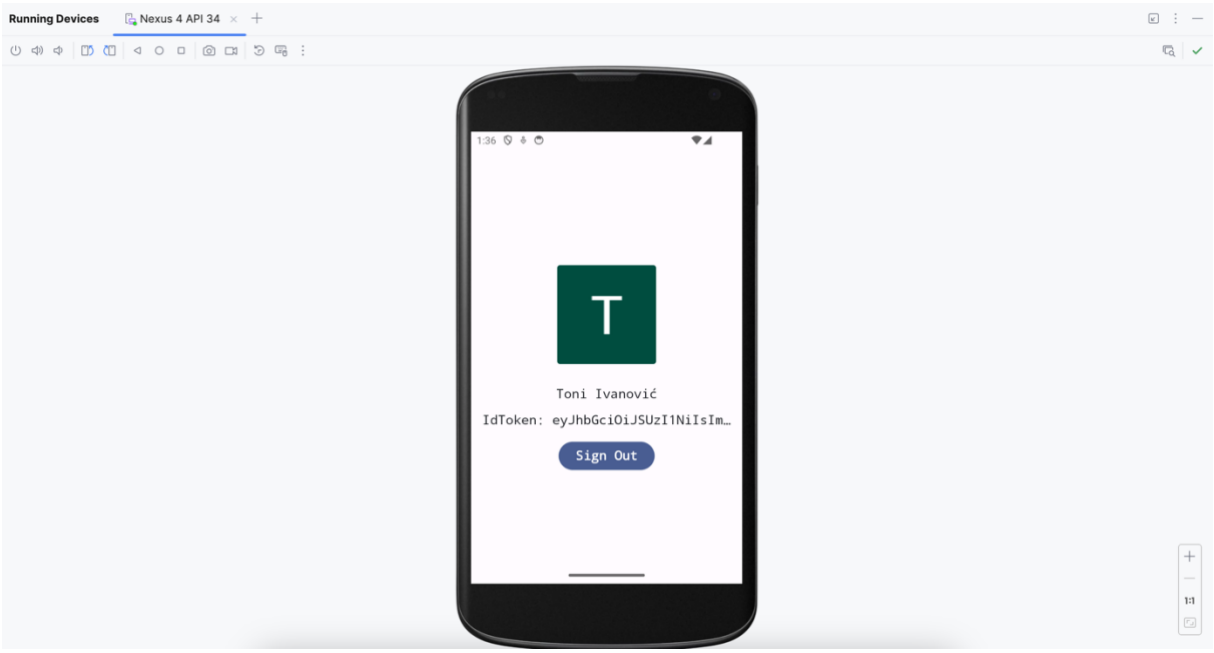
U ovoj klasi sam definirao tri osnovna parametra, id token korisnika, njegovo ime i prezime te sliku profila. Sama logika za prijavu vrlo je slična kao i u prethodnom primjeru budući da se također radi o upotrebi Credential Manager-a, te sam u devetom isječku programskog kôda izdvojio glavne razlike.

```
1. WEB_CLIENT_ID = getString(R.string.default_web_client_id) // - web
   application client id
2. var currentGoogleUser: CurrentGoogleUser = CurrentGoogleUser("", "",
   "".toUri()) // current google user inicijalizacija
3. onGoogleSignInClick = {
4.     val googleIdOption: GetGoogleIdOption =
       GetGoogleIdOption.Builder()
5.         .setServerClientId(WEB_CLIENT_ID)
6.         .setFilterByAuthorizedAccounts(false)
7.         .build()
8.     val request: GetCredentialRequest =
       GetCredentialRequest.Builder()
9.         .addCredentialOption(googleIdOption)
10.        .build()
11.      scope.launch {
12.        try {
13.            val result = credentialManager.getCredential(request =
               request, context = context)
14.            val credential = result.credential
15.            if(credential.type == GoogleIdTokenCredential
                  .TYPE_GOOGLE_ID_TOKEN_CREDENTIAL){
16.                val googleIdTokenCredential =
                    GoogleIdTokenCredential.createFrom(credential.data)
17.                currentGoogleUser = CurrentGoogleUser(
18.                    googleIdToken =
                        googleIdTokenCredential.idToken,
19.                    displayName =
                        googleIdTokenCredential.displayName,
20.                    photoUri =
                        googleIdTokenCredential.profilePictureUri
21.                )
22.                Toast.makeText(context, "Login successful!",
                    Toast.LENGTH_LONG).show()
23.                navController.navigate(Screen.Home.name)
                 }
24.          } catch (e: GoogleIdTokenParsingException){
25.              Toast.makeText(context, "Invalid ID token!",
                  Toast.LENGTH_SHORT).show()
26.          } catch (e: Exception){
```

```
27.                 Toast.makeText(context, "Error: ${e.message}",
                    Toast.LENGTH_LONG).show()
28.             }
29.         }
30.     }
```

Programski kôd 9: Prikaz implementacije procesa prijave u aplikaciju koristeći Google korisnički račun

Dakle kako je prije spomenuto WEB_CLIENT_ID se ovaj puta uzima iz strings.xml dokumenta u koji sam ga sam morao staviti kako bi barem prividno sakrio njegovu vrijednost. Također potrebno je inicijalizirati trenutnog korisnika jer će se isti morati proslijediti u HomeScreen metodu koja se poziva nakon uspješne prijave. Glavna razlika u samom procesu prijave dolazi na liniji 15 gdje se provjerava tip vjerodajnice koji je dobiven te ukoliko on odgovara Google id tokenu iz te vjerodajnice uzimaju se podaci poput imena korisnika, id tokena i njegove slike profila te se prosljeđuju u prije inicijalizirani objekt. Zatim se poziva HomeScreen metoda koja te podatke prikazuje ukoliko su dostupni. Kako izgleda prijavljeni korisnik prikazano je na slici 16.



Slika 16: Prikaz prijavljenog korisnika u aplikaciju putem Google korisničkog računa

Kao što se vidi iz slike početni zaslon aplikacije veoma podsjeća na početni zaslon iz prethodnog poglavlja što je naravno očekivano uzimajući u obzir da se radi o istoj tehnologiji implementiranoj na nešto drugačiji način.

Vrijedno je spomenuti kako je ovaj način implementacije nešto slobodniji uzimajući u obzir neovisnost o Firebase tehnologiji, ali i nešto teži. Kao što sam spomenuo na početku poglavlja današnji preporučeni način za implementaciju Google Sign-in metode jest putem

Credential Manager-a no to se uvijek može promijeniti tako da je na programerima da aktivno prate najnovije pristupe implementaciji što može biti problematično kada se implementira prijava koristeći nekoliko različitih poslužitelja. No kako izgleda implementacija drugog poslužitelja, odnosno prijave putem Facebook računa, prikazat će se u sljedećem poglavlju.

6.2.2. Implementacija prijave putem Facebook računa

Prijava u aplikaciju putem Facebook računa zahtjeva provođenje sličnih koraka kao i kod Google Sign-in metode, ali na nešto drugačiji način. Prvi korak je kreiranje Facebook developer računa kako bi opće imali pristup kreiranju Facebook aplikacije putem koje ćemo autentificirati korisnika. Jednom kada imamo developer vrstu računa potrebno je otići na konzolu za razvojne programere te kreirati novu aplikaciju. Konzola se nalazi na sljedećoj poveznici https://developers.facebook.com/apps/. Ovdje je proces kreacije dosta jednostavan, potrebno je izabrati svrhu aplikacije, odnosno u ovom slučaju autentifikacija putem Facebook login-a, potrebno je naglasiti da se ne radi o izradi igre te dati ime aplikaciji i kontakt email adresu.

Kada je aplikacija kreirana potrebno je pozicionirati se na postavke o toj aplikaciji te izdvojiti najvažnije informacije poput id-a aplikacije i klijentskog tokena. Ove informacije bitne su nam kako bi mogli komunicirati sa aplikacijom. Zatim je potrebno dodati određene ovisnosti u build.gradle.kts te u AndroidManifest.xml dokumente. Točan kôd koji je potrebno dodati u svaki od dokumenata vidljiv je u desetom programskom isječku.

```
1. // build.gradle.kts (:app)
2. implementation("com.facebook.android:facebook-android-
   sdk:latest.release")
3. implementation("com.facebook.android:facebook-login:latest.release")
4.
5. // AndroidManifest.xml
6. <uses-permission android:name="android.permission.INTERNET" />
7.
8. <activity>
9. // ostatak inicijalnog activity kôda
10.   <meta-data
11.     android:name="com.google.android.gms.client_id"
12.     android:value=
         "241995436321-
         q01162bnca6neck4prj0tp9ulahmnraj.apps.googleusercontent.com"
13.   />
14.   <meta-data android:name="com.facebook.sdk.ApplicationId"
        android:value="@string/facebook_app_id"/>
15.   <meta-data android:name="com.facebook.sdk.ClientToken"
        android:value="@string/facebook_client_token"/>
16.  </activity>
17.
18.  <activity android:name="com.facebook.FacebookActivity"
       android:configChanges=
       "keyboard|keyboardHidden|screenLayout|screenSize|orientation"
       android:label="@string/app_name" />
```

```
19.
20.  <activity android:name="com.facebook.CustomTabActivity"
       android:exported="true">
21.    <intent-filter>
22.      <action android:name="android.intent.action.VIEW" />
23.      <category android:name="android.intent.category.DEFAULT" />
24.      <category android:name="android.intent.category.BROWSABLE" />
25.      <data android:scheme="@string/fb_login_protocol_scheme" />
26.    </intent-filter>
27.  </activity>
```

Programski kôd 10: Prikaz implementacije ovisnosti za komunikaciju sa Facebook aplikacijom

Naravno potrebno je ponovno sinkronizirati promjene u projektu te je potrebno spremiti prije izdvojene informacije o id-u aplikacije i klijentskom tokenu u strings.xml dokument. Također potrebno je napraviti i zaseban zapis za shemu login protokola a vrijednost treba biti ista kao i vrijednost id-a aplikacije uz prefiks „fb“. Sada kada je to postavljeno moguće je nastaviti sa postavljanjem novokreirane Facebook aplikacije. Potrebno je još unijeti podatke o imenu paketa te o početnoj klasi što je u ovom slučaju MainActivity. Također nešto što Facebook zahtjeva u ovom procesu jesu sažeci ključeva za razvojnu i objavljenu aplikaciju. Kako generirati ove ključeve vidljivo je u dokumentaciji koja se nalazi u literaturi ovog završnog rada.

S ovime je inicijalno postavljanje postavki za rad Facebook i Android aplikacije gotovo te se može krenuti na pisanje programske logike. Kako bi se zadržala struktura projekta u LoginScreen klasu potrebno je dodati još jedan gumb koji će pozivati onFacebookSignInClick metodu. Ovaj proces implementacije identičan je kao i kod implementacije gumba prilikom prijave sa Google računom. Također kako se ne bi kreirao novi tip korisnika prilikom prijave sa Facebook računom potrebno je preimenovati CurrentGoogleUser klasu u CurrentUser te promijeniti naziv atributa iz googleIdToken u idToken. Ovime se osigurava minimalna promjena implementacije kada se dodaju druga metode prijave u aplikaciju.

Nadalje sve promjene vrše se unutar MainActivity klase. Prvo je potrebno dodati objekt klase CallbackManager koji služi za rad sa odgovorima zahtjeva za prijavu putem Facebook računa. Također potrebno je inicijalizirati FacebookSdk klasu kako bi se komuniciralo sa prije spomenutim SDK-om koji Facebook pruža. Izvorni kôd za implementaciju spomenutog vidljiv je u jedanaestom programskom isječku.

```
1. private lateinit var callbackManager: CallbackManager
2.
3. override fun onCreate(savedInstanceState: Bundle?){
4.     // prijašnji kôd
5.     FacebookSdk.setApplicationId(getString
       (R.string.facebook_app_id ))
6.     FacebookSdk.setClientToken(getString
       (R.string.facebook_client_token))
7.     FacebookSdk.sdkInitialize(this)
```

```
8.      callbackManager = CallbackManager.Factory.create()
9.  }
```

Programski kôd 11: Prikaz implementacije CredentialManager klase i FacebookSDK-a u dosadašnjem kôdu

Unutar ovog kôda važno je napomenuti kako je bitno dodati setApplicationId i setClientToken metode jer ukoliko FacebookSdk prilikom inicijalizacije nema točne podatke o id-u aplikacije i klijentskom tokenu sama prijava neće biti funkcionalna. Idući korak je implementacija onFacebookSignInClick metode. Ona je ovoga puta dosta jednostavnija nego metoda za prijavu putem Google računa, a to je zato što se u njoj isključivo poziva LoginManager klasa koji svoju logiku za rad ima van te metode. Implementacija LoginManager poziva i logike za obradu odgovora vidljiva je u dvanaestom isječku programskog kôda.

```
1.  LoginManager.getInstance()
2.      .registerCallback(callbackManager, object :
        FacebookCallback<LoginResult>{
3.          override fun onCancel() {
4.              Toast.makeText(context, "Facebook login canceled!",
                Toast.LENGTH_SHORT).show()
5.          }
6.
7.          override fun onError(error: FacebookException) {
8.              Toast.makeText(context, "Error: ${error.message}",
                Toast.LENGTH_SHORT).show()
9.          }
10.
11.         override fun onSuccess(result: LoginResult) {
12.             val accessToken = result.accessToken
13.             val request = GraphRequest.newMeRequest(accessToken) {
                obj, _ ->
14.                 try {
15.                     val name = obj?.optString("name", "")
16.                     val profilePictureUri =
                        obj?.getJSONObject("picture")?
                          .getJSONObject("data")?
                          .optString("url", "")
17.                     currentUser = CurrentUser(
18.                         idToken = accessToken.token,
19.                         displayName = name,
20.                         photoUri = profilePictureUri?.toUri()
21.                     )
22.                     navController.navigate(Screen.Home.name)
23.                 } catch (e: JSONException){
24.                     Toast.makeText(context, "Error parsing user
                        data: ${e.message}", Toast.LENGTH_SHORT).show()
25.                 }
26.             }
27.             val parameters = Bundle()
28.             parameters.putString("fields",
                "id,name,picture.type(large)")
29.             request.parameters = parameters
30.             request.executeAsync()
31.         }
32.
33.     })
```

```
34.
35.  onFacebookSignInClick = {
36.      try {
37.          LoginManager.getInstance().logIn(this@MainActivity,
              Arrays.asList("public_profile"))
38.      } catch (e: Exception){
39.          Log.e("Facebook error", "${e.message}")
40.      }
41.  }
```

Programski kôd 12: Prikaz implementacije LoginManager klase za obradu rezultata prijave te metode koja ga poziva

Kao što je već bilo rečeno kada pritisnemo na gumb za prijavu sa Facebook računom samo pozivamo LoginManager klasu unutar koje se prosljeđuje aktivnost iz koje se poziva metoda za prijavu te parametar koji želimo dohvatiti putem te prijave. U ovom slučaju želimo dohvatiti javne informacije o korisničkom profilu te je tako potrebno proslijediti parametar „public_profile“. On je automatski postavljen u Facebook aplikaciji, a kada bi htjeli pristupiti primjerice email podacima bilo bi potrebno podesiti tu mogućnost unutar Facebook aplikacije koju smo prethodno kreirali. No logika za obradu odgovora odvija se unutar LoginManager klase gdje proizvoljno možemo izabrati što želimo napraviti sa korisnikom ovisno o odgovoru koji dobijemo od aplikacije. Ja sam u ovom slučaju izabrao ispisati poruke greške u slučaju da se greška dogodi ili da korisnik odustane od prijave putem Facebook računa. U slučaju da je prijava uspješna prvo se mora izdvojiti pristupni token koji će služiti za pristup informacijama o korisniku. Zatim se kreira novi zahtjev koristeći GraphRequest klasu koja prima pristupni token te pokušava inicijalizirati trenutnog korisnika i postaviti mu informacije poput njegove slike profile, njegovog imena i id tokena. Ukoliko je zahtjev uspješan korisnika se prebacuje na početni zaslon te su tamo vidljive informacije o njemu kao što je to bio slučaj kod prijave putem Google računa.

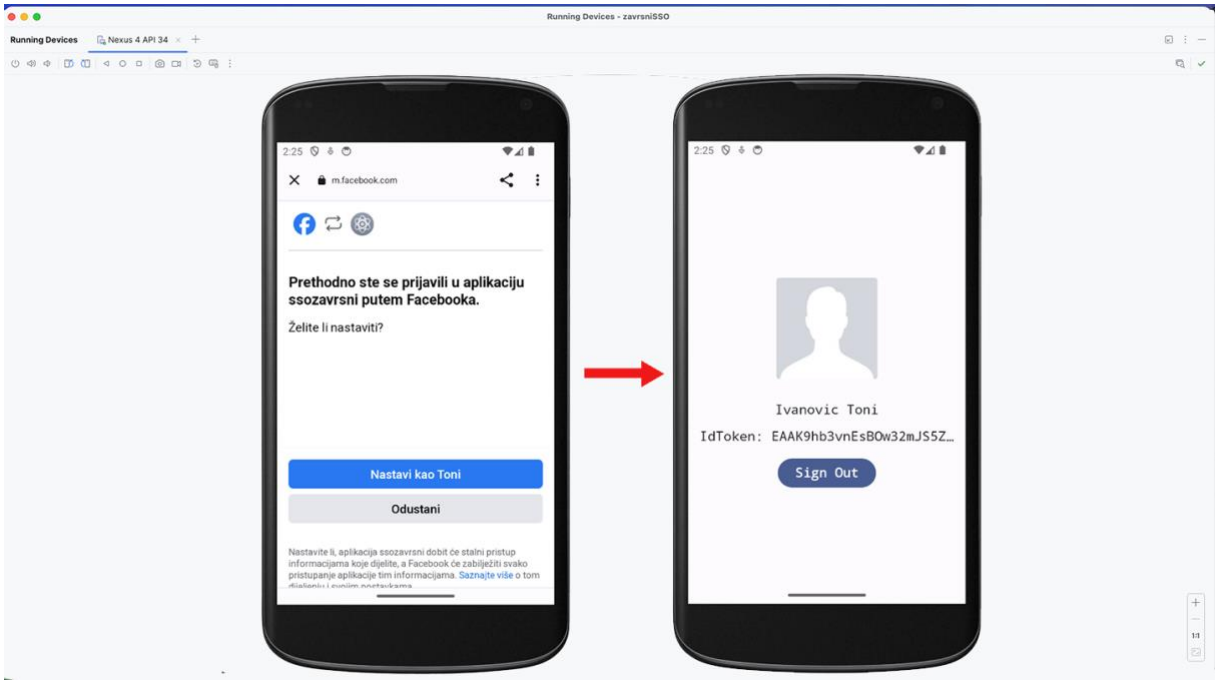
Prije pokretanja aplikacije još je potrebno nadjačati onActivityResult metodu kako bi se u obzir uzimala onActivityResult metoda iz CallbackManager klase. Također ne smije se zaboraviti ni metoda za odjavu iz aplikacije pa je tako potrebno na postojeću logiku dodati i odjavu putem LoginManager klase. Implementacija ovog nadjačanja i metode za odjavu vidljiva je u trinaestom isječku programskog kôda.

```
1. override fun onActivityResult(requestCode: Int, resultCode: Int,
   data: Intent?) {
2.     callbackManager.onActivityResult(requestCode, resultCode, data)
3.     super.onActivityResult(requestCode, resultCode, data)
4. }
5.
6. onSignOutClick = {
7.     scope.launch {
8.         credentialManager.clearCredentialState(
9.             ClearCredentialStateRequest()
```

```
10.          )
11.      }
12.       LoginManager.getInstance().logOut()
13.       navController.popBackStack()
14.       navController.navigate(Screen.Login.name)
15.  }
```

Programski kôd 13: Prikaz implementacije za nadjačanje onActivityResult metode i implementacija odjave

Sada kada je sve postavljeno moguće je pokrenuti aplikaciju i prijaviti se putem Facebook računa. Prvo će biti potrebno prijaviti se u vlastiti Facebook račun, a zatim će biti potrebno odobriti pristup podacima putem Facebook aplikacije koju smo kreirali te ukoliko ne dođe do greške prilikom pristupa podacima korisnik će biti prijavljen. Kako izgleda ovaj proces vidljivo je na slici 17.



Slika 17: Prikaz procesa prijave putem Facebook računa

Kako je vidljivo na slici, jednom kada se korisnik prijavi u Facebook račun i da dopuštenje aplikaciji koju smo kreirali ranije da koristi naše podatke korisnik je prijavljen i njegovi podaci vidljivi su na početnom zaslonu.

Dakle kako je vidljivo kroz cijelo poglavlje, iako sam proces implementacije nije isti kao kod prijave putem Google računa, rezultat je isti. Nešto više o samim razlikama biti će rečeno u sljedećem potpoglavlju.

6.2.3. Osvrt na SSO implementacije

Kroz prethodna poglavlja prikazana su dva načina za obavljanje iste funkcionalnosti, odnosno prijave u aplikaciju putem SSO pristupa prijavi. Kako je i vidljivo iako su rezultati na kraju isti, proces dolaska do rezultata može se znatno razlikovati. Tako sam u trećoj tablici ovog završnog rada usporedio ove dvije tehnologije.

Tablica 3: Sistematizacija sličnosti i razlika u SSO tehnologijama

Kriterij	Google Sign-in	Facebook login
Jednostavnost implementacije	Jednostavna ali podložna češćim promjenama	Složenija implementacija zbog ovisnosti o FacebookSDK-u, ali standardna već dugi niz godina
Kvaliteta dostupne dokumentacije (1-5)	3	4
Sigurnost i privatnost	Visoka	Visoka
Korišteni protokoli	OAuth 2.0, OIDC	OAuth 2.0, OIDC
Ovisnost	Ovisi o Google uslugama	Ovisi o Facebook uslugama
Podržane platforme	Android, iOS, Web, stolne aplikacije i ostali uređaji (npr. smart TV)	Android, iOS, Web, stolne aplikacije i ostali uređaji (npr. smart TV)

Kako je i iz same tablice vidljivo ova dva pristupa iznimno su slična međusobno, a jedina velika razlika jest u samoj implementaciji. Zato je važno da prije nego se odlučimo za integraciju nekog od rješenja dobro informiramo o ovisnostima koje implementacija tih rješenja zahtjeva kako bi mogli dobro odabrati tehnologije koje želimo koristiti u našim projektima. Osobno preferiram Google Sign-in implementaciju jer iako dokumentacija za implementaciju putem Credential Manager-a još uvijek nije na nivou na kojem smatram da bi trebala biti i dalje se koriste relevantne i ažurne metode. Nasuprot tome je Facebook login koji po mom mišljenju ima kvalitetnije pisanu i pristupačniju dokumentaciju, ali ona sama nije ažurirana posebno za najnovije tehnologije te može doći do potrebe za korištenjem zastarjelih (eng. *deprecated*) metoda što definitivno nije preporučljivo posebno u produkcijskom kôdu.

6.3. Implementacija Auth0 gotovog rješenja

Prema svojoj zadaći, Auth0 slično je rješenje kao i Firebase Authentication uzimajući u obzir da pruža eksternalizaciju procesa prijave ili registracije u aplikaciju putem vlastite platforme. Ipak po čemu se razlikuje od Firebase Authentication principa je po količini raznih metoda koje izvorno podržava. Tako će u ovom poglavlju biti opisan proces implementacije registracije i prijave u aplikaciju koristeći standardnu kombinaciju email adrese i lozinke, Google i Facebook računa te višerazinske autentifikacije koristeći OTP princip.

Za implementaciju Auth0 rješenja potrebno je imati već postojeću aplikaciju koja će služiti kao temelj za integraciju pa je ista kreirana prema opisu koji se nalazi na vrhu poglavlja o implementaciji, ali je moguće i preuzeti predložak koji sami dizajniramo putem njihove Web aplikacije. U nadolazećim potpoglavljima opisati ću proces kreiranja Auth0 aplikacije i njene integracije u Android aplikaciju.

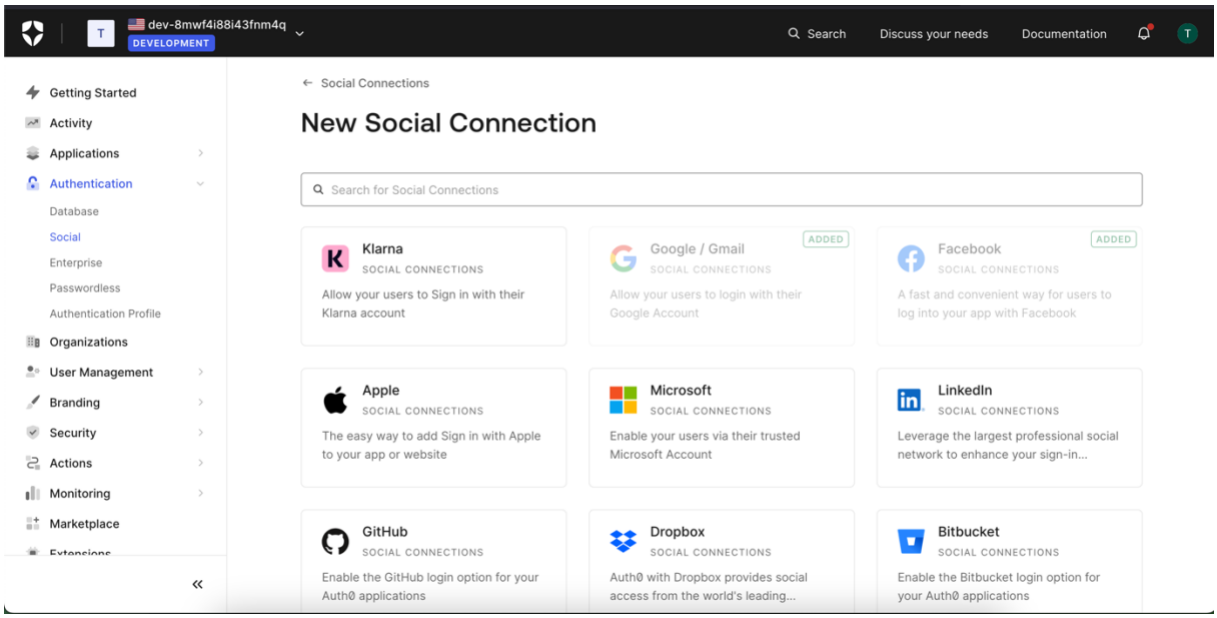
6.3.1. Kreiranje i postavljanje nove Auth0 aplikacije

Jednom kada imamo kreiranu Android aplikaciju možemo započeti sa kreiranjem Auth0 aplikacije putem njihove Web aplikacije koja se nalazi na poveznici https://manage.auth0.com/.

Prvenstveno je potrebno kreirati korisnički račun putem kojeg će se pristupati Auth0 platformi, a zatim je moguće preći na kreiranje vlastite aplikacije. U Auth0 konzoli potrebno se je pozicionirati na sekciju za aplikacije (eng. *applications*) te kliknuti na gumb kreiraj aplikaciju (eng. *create application*). Potrebno je izabrati izvorni tip aplikacije te joj dati proizvoljno ime. Nakon toga najjednostavniji princip za implementaciju je praćenje uputa unutar sekcije za brzo postavljanje (eng. *quickstart*) (Auth0, bez dat.-a).

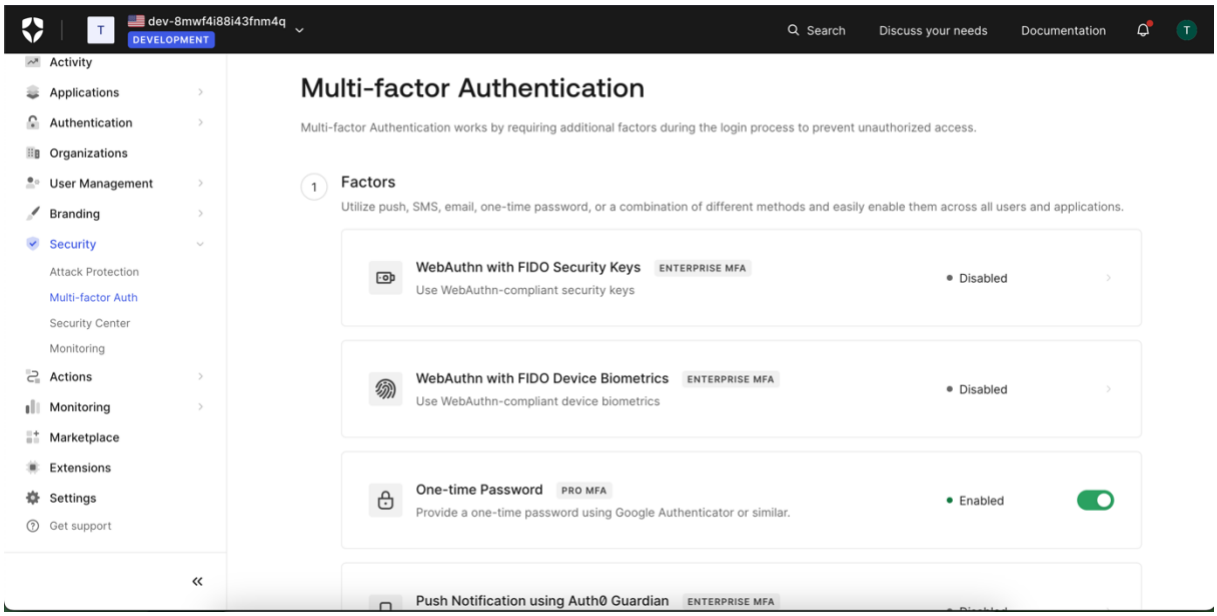
Tako prije nego se krene u integraciju aplikacije sa Android aplikacijom potrebno je izdvojiti Auth0 domenu (eng. *domain*) i klijentski id vjerodajnice. Putem njih komunicirati će se sa Auth0 aplikacijom. Sljedeći nužan korak u postavljaju Auth0 aplikacije je definiranje putanja za povratne pozive (eng. *callback*), a kako se radi o aplikaciji namijenjenoj za demonstraciju ja ću koristiti putanje koje Auth0 platforma nudi za aplikacije te namjene, a dostupni su putem sekcije za brzo postavljanje. Zadnji korak koji je specifičan za implementaciju ove aplikacije jest odabir SSO poslužitelja te postavljanje višerazinske autentifikacije.

Na slici 18 prikazana je sekcija putem koje se odabiru sve socijalne konekcije koje želimo pružati kao mogućnosti za autentifikaciju i autorizaciju korisnika. Ja sam odabrao specifično Google i Facebook platforme.



Slika 18: Prikaz Auth0 Web aplikacije i sekcije za odabir SSO poslužitelja

Nadalje se je potrebno prebaciti u odjeljak za sigurnost (eng. *security*) i kao što je vidljivo na slici 19 uključiti višerazinsku autentifikaciju.



Slika 19: Prikaz Auth0 platforme i sekcije za uključivanje višerazinske autentifikacije

Još je dobro provjeriti da je MFA opcija uključena da uvijek traži korisnika da se prijavi koristeći jednokratnu lozinku te se može krenuti na integraciju ove aplikacije u Android aplikaciju.

6.3.2. Integracija Auth0 aplikacije s Android aplikacijom

Kao i do sada osnovna struktura aplikacije je ista, tako se aplikacija sastoji od MainActivity, LoginScreen, HomeScreen i CurrentUser klasa.

Prvi korak je dodavanje ovisnosti u build.gradle i AndroidManifest dokumente. Ovisnosti koje je potrebno dodati u build.gradle dokument vidljive su u četrnaestom isječku programskog kôda.

```
1. // build.gradle(:app)
2. // default config odjeljak
3. manifestPlaceholders["auth0Domain"] = "@string/auth_0_domain"
4. manifestPlaceholders["auth0Scheme"] = "demo"
5.
6. // dependencies odjeljak
7. implementation("androidx.navigation:navigation-compose:2.7.7") //
   biblioteka za navigaciju
8. implementation("io.coil-kt:coil-compose:2.6.0") // biblioteka za
   učitavanje slika
9. implementation("com.auth0.android:auth0:2.+") // auth0 biblioteka
```

Programski kôd 14: Prikaz ovisnosti u build.gradle dokumentu za auth0 implementaciju

Ono što je vidljivo drugačije od svakog uključivanja ovisnosti do sada jesu manifestPlaceholders vrijednosti. One se koriste kako bi se specificirala konfiguracija prilikom pokretanja aplikacije, a za rad sa Auth0 aplikacijom potrebno je definirati shemu i domenu. Domena je jednako kao i klijentski id spremljena u strings.xml dokument kako ne bi bila direktno čitljiva iz programskog kôda. U AndroidManifest dokument potrebno je dodati dopuštenje za korištenje interneta, a kako to izgleda bilo je već prikazano u desetom programskom isječku.

Sada kada je projekt postavljen i sinkroniziran može se krenuti sa implementacijom programske logike. Klase LoginScreen, HomeScreen i CurrentUser iznimno su slične ili iste prema svojoj implementaciji prikazanoj u prijašnjim primjerima te se njihova implementacija ovdje neće ponavljati. Tako je prva razlika u MainActivity klasi. Potrebno je inicijalizirati varijablu klase Auth0 te mu proslijediti valjane parametre, odnosno klijentski id i domenu. Inicijalizacija klase vidljiva je u isječku programskog kôda 15.

```
1. private lateinit var account: Auth0
2.
3. account = Auth0(
4.     getString(R.string.auth_0_client_id), // klijentski id
5.     getString(R.string.auth_0_domain) // domena
6. )
```

Programski kôd 15: Prikaz implementacije Auth0 klase i njene inicijalizacije sa potrebnim vrijednostima

Kada je klasa inicijalizirana sve što je potrebno za implementaciju prijave i odjave jest WebAuthProvider klasa iz auth0 biblioteke. Osnovna implementacija metode za prijavu vidljiva je u šesnaestom isječku programskog kôda.

```
onSignInClick = {
    WebAuthProvider.login(account)
        .withScheme("demo")
        .withScope("openid profile name")
        .start(context, object : Callback<Credentials,
         AuthenticationException>{
            override fun onFailure(error: AuthenticationException) {
                Toast.makeText(context, "Authentication error:
                 ${error.message}", Toast.LENGTH_SHORT).show()
            }

              override fun onSuccess(result: Credentials) {
                  /* TODO */
              }

         })
}
```

Programski kôd 16: Prikaz osnovne implementacije metode za prijavu u aplikaciju

Kao što je iz programskog kôda vidljivo poziva se WebAuthProvider klasa sa svojom login metodom unutar koje se prosljeđuje varijabla account koja sadrži vjerodajnice za našu aplikaciju. Nužno je definirati shemu i opseg ove metode te je tako ovdje korištena ista shema kao i u build.gradle dokumentu, a opseg je onaj koji preporuča dokumentacija za razvojne programere. Kada se metoda pokrene dobivaju se dva stanja, stanje uspjeha ili neuspjeha. Stanje neuspjeha ovdje samo daje poruku o grešci, a kako izgleda implementacija za uspješnu prijavu vidljivo je u sedamnaestom programskom isječku.

```
override fun onSuccess(result: Credentials) {
    val accessToken = result.accessToken
    val client = AuthenticationAPIClient(account)

    client.userInfo(accessToken)
        .start(object : Callback<UserProfile,
         AuthenticationException>{
            override fun onFailure(error: AuthenticationException) {
                Toast.makeText(context, "Error getting user data:
                 ${error.message}", Toast.LENGTH_SHORT).show()
            }

              override fun onSuccess(result: UserProfile) {
                  val name = result.name
                  val photoUri = result.pictureURL
                  currentUser = CurrentUser(
                      idToken = accessToken,
                      displayName = name,
                      photoUri = photoUri?.toUri()
                  )
```

```
                    navController.navigate(Screen.Home.name)
                }

            })
    }
```

Programski kôd 17: Prikaz implementacije za uspješnu prijavu

Ako je prijava uspješna vratiti će određene vjerodajnice u svom rezultatu. Tako se ovdje pristupni token može izdvojiti iz rezultata, a isti će se koristiti u AuthenticationAPIClient klasi za pristup podacima o korisniku. Ova klasa također ima stanje uspjeha i neuspjeha te je neuspjeh implementiran kao i prije, a ukoliko se uspješno dohvate podaci o korisniku kreira se novi objekt klase CurrentUser sa podacima poput imena i slike profile te se korisnika preusmjerava na početni zaslon.

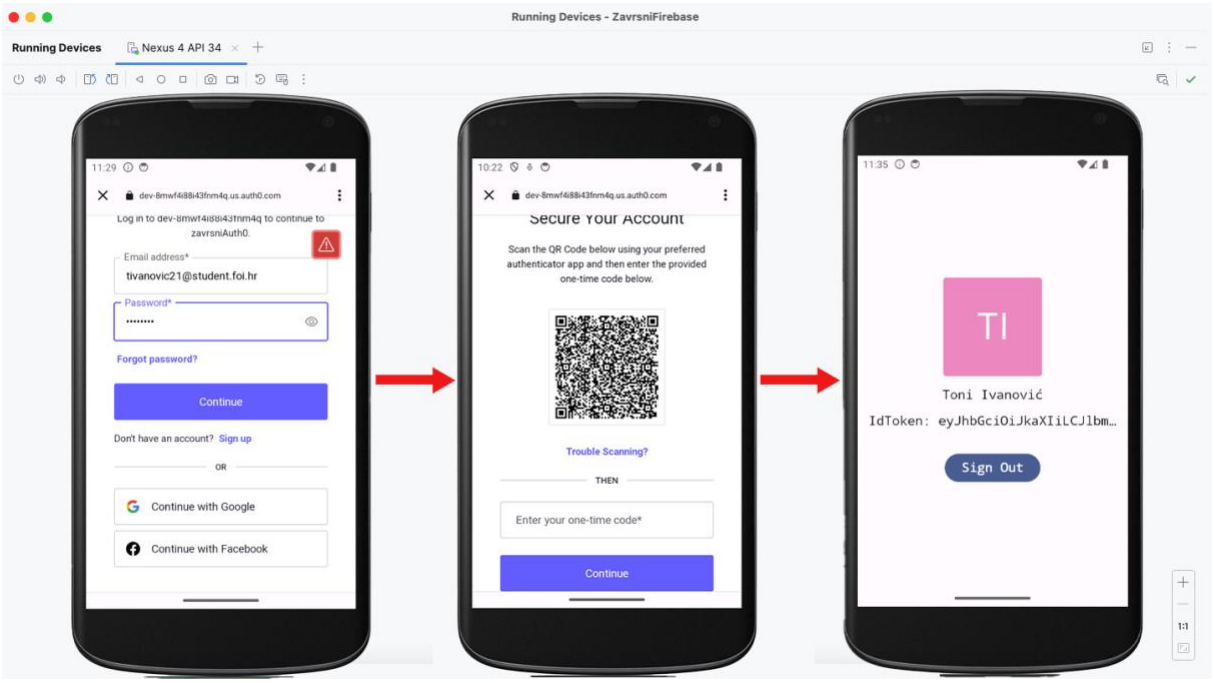
Cijeli proces prijave tako se odvija putem Web preglednika na mobitelu koji poziva aplikaciju koju smo ranije kreirali. Unutar te aplikacije ponuđene su sve one opcije koje smo sami izabrali, te je tako moguće stvoriti novi račun za prijavu u aplikaciju, prijaviti se sa postojećim ili se prijaviti putem Google ili Facebook računa. Prije nego što pokrenemo aplikaciju potrebno je implementirati i odjavu pa je tako implementacija programske logike za odjavu vidljiva u osamnaestom isječku programskog kôda.

```
onSignOutClick = {
    WebAuthProvider.logout(account)
        .withScheme("demo")
        .start(context, object : Callback<Void?,
         AuthenticationException>{
            override fun onFailure(error: AuthenticationException) {
                Toast.makeText(context, "Error while signing out:
                 ${error.message}", Toast.LENGTH_SHORT).show()
            }

            override fun onSuccess(result: Void?) {
                  navController.navigate(Screen.Login.name)
              }
         })
}
```

Jednako kao i prije, poziva se WebAuthProvider klasa sa pripadajućom logout metodom koja vraća stanje uspjeha i neuspjeha. Slučaj neuspjeha implementiran je kao i ranije, a u slučaju uspješne odjave vraća se korisnika na zaslon za prijavu.

Na slici 20 vidljiv je proces prijave u aplikaciju koristeći račun kreiran preko Auth0 aplikacije.



Slika 20: Prikaz procesa prijave u aplikaciju koristeći račun kreiran putem platforme Auth0

Kako je na slici prikazano korišteni su podaci za prijavu koji su dobiveni putem registracije korisničkog računa preko Auth0 platforme. Kako je ranije postavljeno da je višerazinska autentifikacija obavezna tako je u ovom procesu bilo potrebno skenirati QR kôd sa nekom od aplikacija koje nude mogućnosti generiranja jednokratnih lozinki, poput Google Authenticator aplikacije te se na njoj generira kôd od šest znamenki koji se koristi za autentifikaciju. Kada prijava uspije korisnik je preusmjeren na početni zaslon koji je vidljiv na trećem mobitelu na slici. Pritiskom gumba za odjavu također se pokreće Web preglednik te se korisnika odjavljuje iz Auth0 aplikacije, a potom i iz Android aplikacije.

Procesi prijave putem Google i Facebook računa identični su kao i u prošlim implementacijama uz iznimku da i oni zahtijevaju korištenje jednokratne lozinke, a još se može spomenuti da iako sam koristio istu email adresu za kreiranje korisničkog računa putem platforme i za prijavu putem postojećeg Google računa oni nisu povezani te iako dijele email adresu imaju različite podatke kao i različite jednokratne lozinke.

Za sve daljnje promjene u ovom procesu potrebno je pristupiti Auth0 Web aplikaciji putem koje se može modificirati izgled samo sučelja, dodavati dodatne metode autentifikacije, brisati ili ograničavati korisnici i još puno toga.

6.3.3. Osvrt na Auth0 tehnologiju

Iz prethodnih poglavlja vidljivo je kako je sam proces implementacije ove tehnologije zapravo dosta jednostavan te nudi najviše mogućnosti. Iako ima svojih mana kao što je potpuna ovisnost o Auth0 platformi prema svojoj implementaciji čini se kao najjednostavniji

način za implementaciju nešto kompleksnijih i raznolikijih metoda prijave i registracije u aplikaciju. Brojne metode nisu ni spomenute u poglavlju jer nisu dostupne u okviru besplatnog projekta, ali Auth0 nudi razne metode višerazinske autentifikacije, biometrijsku autentifikaciju, autentifikaciju putem personaliziranih SAML, OIDC i LDAP protokola i još pregršt metoda.

Sve u svemu Auth0 platforma nudi jednostavno i efektivno rješenje, posebno ako se radi o organizaciji koja ima raznolike potrebe, ali ovo rješenje može biti previše kada se radi o manjim ili srednjim projektima.

6.4. Osvrt na implementacije svih tehnologija

Kroz cijelo poglavlje o implementaciji trudio sam se prikazati najjednostavniji način za integraciju tehnologija sa postojećim Android aplikacijama. Kada sam radio samu implementaciju pojedinih tehnologija bilo je raznih izazova što zbog zastarjele dokumentacije, što zbog drugačijeg pristupa razvoju same aplikacije, ali sam se držao istog principa u svakoj implementaciji kako bi pokazao tehnologije koje je moguće implementirati u aplikacije koje su pisane prema današnjim standardima.

Android kao platforma je vrlo promjenjiva, posebno kada se govori o primjeni novih tehnologija poput Jetpack Compose-a i Kotlin programskog jezika pa je tako neke tehnologije iznimno teško primijeniti. Konkretno prilikom implementacije prijave putem Google i Facebook računa, odnosno pri pisanju 6.2. poglavlja nailazio sam na razne probleme zbog loše dokumentacije ili jednostavno zbog potrebe za odstupanjem od iste.

Tako mogu reći da iako svaki način integracije ima svoje prednosti i svoje mane, ukoliko se radi o projektu koji nije na razini poduzeća, okrenuo bih se ka korištenju Firebase platforme, a posebno ako se radi o srednjim ili manjim projektima, a ukoliko se radi o poduzećima mislim da Auth0 platforma definitivno služi kao bolji odabir od vlastite implementacije. Naravno sve ovisi o potrebama aplikacije i naravno rješenje koje sami napravimo biti će najbolje jer će biti prilagođeno našim potrebama, ali ako gledam u okviru uporabe trećih strana za obavljanje iznimno važnih, ali standardiziranih procesa registracije i prijave u aplikaciju okrenuo bih se na dvije gore spomenute opcije.

7. Zaključak

Kroz ovaj rad prošao sam puno tema, od protokola i standarda na kojima rade nama dobro poznati procesi registracije i prijave, tako i do tehnologija koje te procese omogućuju. Kada sam pisao rad pokušao sam se staviti u cipele prosječne osobe, nekoga tko vjerojatno razumije osnovne koncepte registracije ili prijave u aplikaciju barem na nekoj kolokvijalnoj razini, ali i osobe koja uz to ima interes za razumijevanje tih procesa u njihovu dubinu, ispod samih osnovnih interakcija sa grafičkim sučeljem.

Tako sam kroz poglavlje o protokolima, posebno kada je riječ o OAuth 2.0 protokolu pokušao prikazati odnose iza svakog koraka koji se dogodi kada mi kao korisnik zatražimo pristup aplikaciji. Upravo zbog toga je bilo puno usporedbi sa svakodnevnim životom i možda ponekad nepotrebnog pojednostavljivanja samih procesa, ali cilj mi je bio iste približiti svakome tko se odluči na čitanje ovog rada. Smatram da je vrlo korisno znati kako rade stvari koje koristim svakodnevno jer većina stvari ne funkcionira na neki neponovljiv način, već se dosta protokola i obrazaca ponavlja na više mjesta, neki u digitalnom svijetu, a neki i u onom stvarnom. Danas je OAuth 2.0 svuda oko nas, gotovo svaka aplikacija zahtjeva njegovu primjenu te se nadam da sam kroz ovaj rad uspio objasniti zašto je tome tako, odnosno da sam uspio približiti njegovu svrhu i primjenu, posebno u odnosu na druge spomenute protokole.

Poglavlje o tehnologijama je tu ponajviše zbog toga što je jako lijepo znati stvari u teoriji, ali ih je ipak potrebno primijeniti kako bi samo znanje imalo smisla. Kada sam izučavao koje tehnologije odabrati za ovaj završi rad, jer ih je stvarno pregršt dostupno, trudio sam se ići putem moderne upotrebe, odnosno birao sam one tehnologije koje su danas standardne. Upravo iz tih razloga birao sam Firebase i Auth0 platforme te Google i Facebook kao primarne autorizacijske poslužitelje jer je to ono što većina današnjih aplikacija koristi i to je nešto sa čime se imamo priliku najviše susresti. Također tehnologije za razvoj Android aplikacija, odnosno Jetpack Compose i Kotlin programski jezik, izabrane su zato što su to tehnologije koje se danas zagovara, tehnologije koja za cilj imaju jednog dana zamijeniti XML način pisanja dizajna te pisanje Android aplikacija u Java programskom jeziku. Iako je ponekad bilo problema prilikom implementacije zbog korištenja ovih tehnologija i dalje mislim da je odabir bio valjan jer sam ovaj rad pisao u nadi da će i u ne toliko bliskoj budućnosti imati svoju važnost i težinu.

U poglavlju o implementaciji izabrao sam ići putem implementacije koji će prikazati uporabu svega što je prije spomenuto. Tako sam se trudio ne ići van okvira odabranih

tehnologija, jer iako je moguće, pokušao sam najbolje moguće prikazati same tehnologije i njihove mogućnosti kako bi jasno iskazao što neka tehnologija može poduprijeti, a što ne. Jer iako sve tehnologije služe kao svojevrsna nadogradnja na već postojeće aplikacije razlikuju se u svojim mogućnostima te se nadam da se uspio sistematizirati i podijeliti ih prema njihovim sposobnostima.

Na kraju smatram da je osnovni cilj rada ostvaren. Smatram da su danas korišteni protokoli adekvatno sumirani i sistematizirani, da su prikazane i opisane danas korištene tehnologije te da poglavlje o implementaciji pruža adekvatne principe za implementaciju svega spomenutog. No najvažnije od svega, nadam se da je ovaj rad uspio približiti apstraktne procese koje uglavnom vidimo samo u digitalnom svijetu svima koji su odrasli u fizičkom svijetu, a život ih je kasnije doveo i u ovo digitalno okruženje.

Popis literature

Anderson, R. (2008). *Security engineering: A guide to building dependable distributed systems* (2. ed). Indianapolis, Ind: Wiley.

Android—Facebook Login—Documentation. (bez dat.). Preuzeto 25. svibanj 2024., od Meta for Developers website: https://developers.facebook.com/docs/facebook-login/android/

Auth0. (bez dat.-a). Auth0 Native/Mobile App Quickstarts. Preuzeto 04. lipanj 2024., od Auth0 Docs website: https://auth0.com/docs/

Auth0. (bez dat.-b). Auth0 Overview. Preuzeto 25. svibanj 2024., od Auth0 Docs website: https://auth0.com/docs/

Auth0. (bez dat.-c). Enterprise Identity Providers. Preuzeto 25. svibanj 2024., od Auth0 Docs website: https://auth0.com/docs/

Authenticate with Google on Android | Firebase Authentication. (bez dat.). Preuzeto 30. svibanj 2024., od Firebase website: https://firebase.google.com/docs/auth/android/google-signin

Chen, E., Tian, Y., Pei, Y., Kotcher, R., Chen, S., & Tague, P. (2014, studeni 1). *OAuth Demystified for Mobile Application Developers*. Predstavljeno na Proceedings of the ACM Conference on Computer and Communications Security. https://doi.org/10.1145/2660267.2660323

Clarke, N., Furnell, S., & Reynolds, P. (2002). Biometric Authentication for Mobile Devices. *Proceeding of the 3rd Australian Information Warfare and Security Conference*.

de Medeiros, B., Agarwal, N., Sakimura, N., Bradley, J., & Jones, M. B. (2014). Final: OpenID Connect Core 1.0. Preuzeto 23. svibanj 2024., od https://openid.net/specs/openid-connect-core-1_0-final.html

Firebase Authentication. (bez dat.). Preuzeto 25. svibanj 2024., od Firebase website: https://firebase.google.com/docs/auth

Griffiths, D., & Griffiths, D. (2021). *Head first Android development* (Third edition). Beijing Boston Farnham Sebastopol Tokyo: O'Reilly.

Hardt, D. (2012). *The OAuth 2.0 Authorization Framework* (Request for Comments Izd. RFC 6749). Internet Engineering Task Force. https://doi.org/10.17487/RFC6749

Hrvatski jezični portal. (bez dat.). Preuzeto 18. svibanj 2024., od https://hjp.znanje.hr/index.php?show=search_by_id&id=dlpiXBk%3D

Hughes, J., & Maler, E. (2005). Security assertion markup language (saml) v2. 0 technical overview. *OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08*, *13*, 12.

Integrate Credential Manager with Sign in with Google. (bez dat.). Preuzeto 22. svibanj 2024., od Android Developers website: https://developer.android.com/training/sign-in/credential-manager

Jones, M. B., & Hardt, D. (2012). *The OAuth 2.0 Authorization Framework: Bearer Token Usage* (Request for Comments Izd. RFC 6750). Internet Engineering Task Force. https://doi.org/10.17487/RFC6750

Kermek, D., Novak, M., Kaniški, M., & Levak, I. (2022). *Priručnik za predmete Razvoj Web aplikacija (ITDP), Razvoj Web aplikacija (IPS)*. Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin.

Li, W. (2017). *Improving the Security of Real World Identity Management Systems* (PhD Thesis). Royal Holloway, University of London.

McDonald, M. (2020). *Web security for developers: Real threats, practical defense*. San Francisco: No Starch Press.

Mehta, B. (2014). *RESTful Java patterns and best practices: Learn best practices to efficiently build scalable, reliable, and maintenable high performance RESTful services*. Birmingham: Packt Publishing.

Prlenda, M. (2018). *Biometrijska identifikacija putnika u zaštiti zračnog prometa* (str. 41) [Završni rad]. Zagreb: Sveučilište u Zagrebu, Fakultet prometnih znanosti. Preuzeto od Sveučilište u Zagrebu, Fakultet prometnih znanosti website: https://urn.nsk.hr/urn:nbn:hr:119:014879

Sakimura, N., Bradley, J., & Agarwal, N. (2015). *Proof Key for Code Exchange by OAuth Public Clients* (Request for Comments Izd. RFC 7636). Internet Engineering Task Force. https://doi.org/10.17487/RFC7636

Schneier, B. (2006). *Beyond fear: Thinking sensibly about security in an uncertain world* (Repr). New York, NY: Copernicus Books.

STANISLAV, M. (2015). *Two-Factor Authentication*. IT Governance Publishing. JSTOR. https://doi.org/10.2307/j.ctt15hvwnz

Popis slika

Slika 1: Prikaz koraka komunikacije OAuth protokola tijekom autorizacije (Hardt, 2012, Poglavlje 1.2) 11
Slika 2: Prikaz toka tijekom korištenja pristupnog i osvježavajućeg tokena (Hardt, 2012, Poglavlje 1.5) 15
Slika 3: Prikaz osnovnog SAML toka (Mehta, 2014, Poglavlje 3) 21
Slika 4: Početni zaslon prilikom kreiranja novog projekta u Android Studiju 28
Slika 5: Prikaz zaslona za imenovanje projekta u Android Studiju 29
Slika 6: Prikaz procesa za dobivanje SHA-1 sažetka aplikacije putem Android Studija 30
Slika 7: Prikaz Firebase konzole sa odabranom metodom za autentifikaciju 31
Slika 8: Prikaz Firebase konzole za dodavanje novog korisnika 32
Slika 9: Početni zaslon nakon uspješne prijave 35
Slika 10: Prikaz Firebase konzole za dodavanje Google Sign-in mogućnosti 35
Slika 11: Prikaz procesa za dodavanje nove klase u Android Studiju 37
Slika 12: Prikaz procesa prijave u aplikaciju koristeći Google račun putem Firebase-a 43
Slika 13: Prikaz početnog zaslona nakon prijave u aplikaciju koristeći email adresu i lozinku putem Firebase-a 43
Slika 14: Prikaz putanje do OAuth Consent Screen dijela Google konzole 45
Slika 15: Prikaz postavljene OAuth aplikacije putem Google Cloud konzole 46
Slika 16: Prikaz prijavljenog korisnika u aplikaciju putem Google korisničkog računa 48
Slika 17: Prikaz procesa prijave putem Facebook računa 53
Slika 18: Prikaz Auth0 Web aplikacije i sekcije za odabir SSO poslužitelja 56
Slika 19: Prikaz Auth0 platforme i sekcije za uključivanje višerazinske autentifikacije 56
Slika 20: Prikaz procesa prijave u aplikaciju koristeći račun kreiran putem platforme Auth0 . 60

Popis tablica

Tablica 1: Prikaz atributa ID tokena (de Medeiros, Agarwal, Sakimura, Bradley, & Jones, 2014, Poglavlje 2)..... 19
Tablica 2: Sistematizacija tehnologija za proces autentifikacije u aplikacijama 26
Tablica 3: Sistematizacija sličnosti i razlika u SSO tehnologijama 54

Popis programskih kôdova

Programski kôd 1: MainActivity dodavanje zaslona za autentifikaciju 33
Programski kôd 2: AuthScreen metoda za imlpementaciju prijave i registracije 34
Programski kôd 3: Ovisnosti za Credential Manager i Google Sign-in mogućnost autentifikacije 36
Programski kôd 4: Osnovna implementacija LoginScreen metode 37
Programski kôd 5: Poziv metode za prijavu putem email adrese i lozinke u LoginScreen metodi 38
Programski kôd 6: Implementacija HomeScreen klase uz primjer dohvaćanja slike profila korisnika 39
Programski kôd 7: Prikaz MainActivity klase koja implementira logiku za prijavu i registraciju putem email adrese i lozinke te prijavu putem Google računa 41
Programski kôd 8: Implementacija data klase za prijavljenog korisnika putem Google računa 47
Programski kôd 9: Prikaz implementacije procesa prijave u aplikaciju koristeći Google korisnički račun 48
Programski kôd 10: Prikaz implementacije ovisnosti za komunikaciju sa Facebook aplikacijom 50
Programski kôd 11: Prikaz implementacije CredentialManager klase i FacebookSDK-a u dosadašnjem kôdu 51
Programski kôd 12: Prikaz implementacije LoginManager klase za obradu rezultata prijave te metode koja ga poziva 52
Programski kôd 13: Prikaz implementacije za nadjačanje onActivityResult metode i implementacija odjave 53
Programski kôd 14: Prikaz ovisnosti u build.gradle dokumentu za auth0 implementaciju 57
Programski kôd 15: Prikaz implementacije Auth0 klase i njene inicijalizacije sa potrebnim vrijednostima 57
Programski kôd 16: Prikaz osnovne implementacije metode za prijavu u aplikaciju 58
Programski kôd 17: Prikaz implementacije za uspješnu prijavu 59