Notifikacije
Uvod u Firebase

prof. dr Bratislav Predić
dipl. inž. Nevena Tufegdžić

Razvoj mobilnih aplikacija i servisa
Računarstvo i informatika

2024.

Notifikacije u Androidu

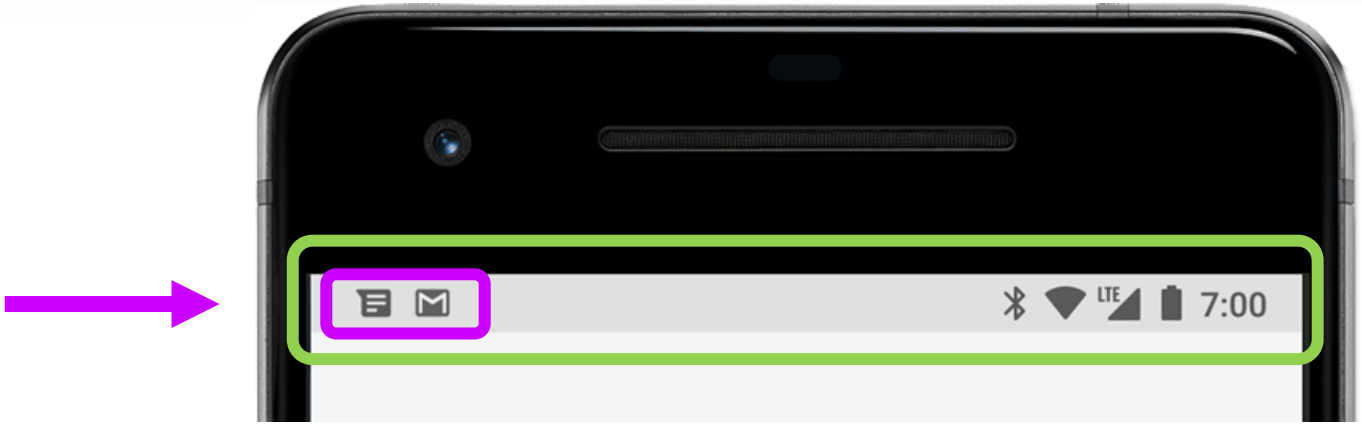
- Notifikacija predstavlja poruku koju Android prikazuje van UI-ja aplikacije
- Aplikacija na ovaj način obaveštava korisnika o nečemu
- Korisnik može izvršiti tap na notifikaciju kako bi otvorio aplikaciju, ili izvršiti neke akcije koje sama notifikacija nudi (reply, mute, itd)

Prikaz notifikacija

- Najčešće lokacije prikazivanja notifikacija su:
  - Status bar
  - Notification drawer
  - Heads-up notifikacija u floating window-u

Status bar i notification drawer

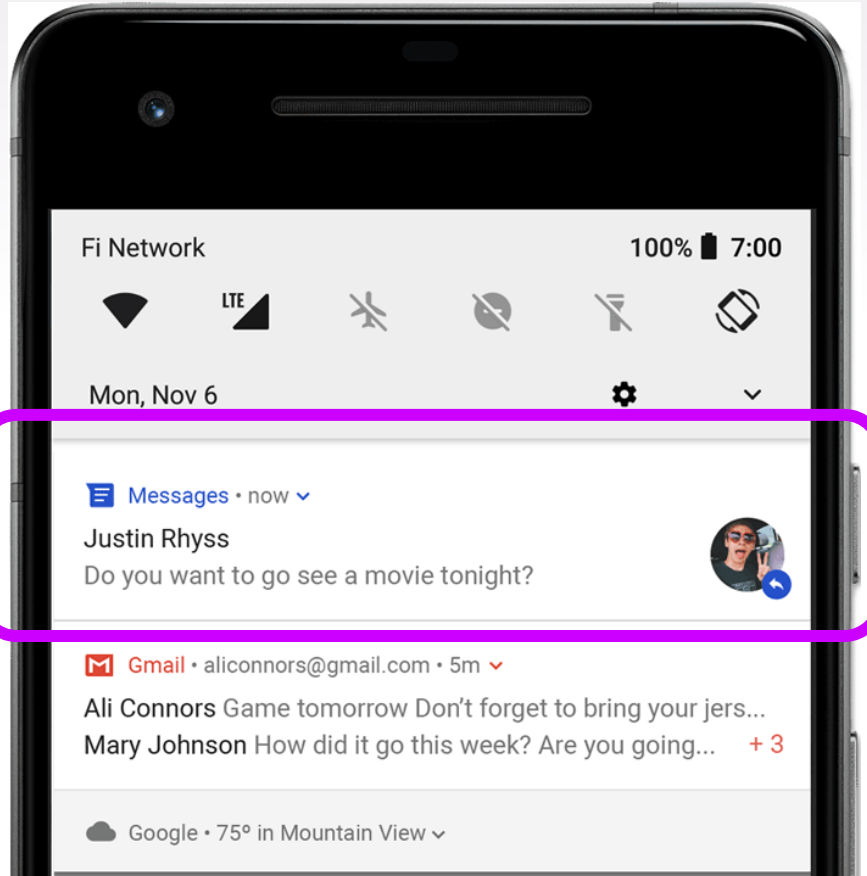
- Kada sistem primi notifikaciju, prikazuje je u okviru status bar-a u vidu ikonice



Status bar i notification drawer

- Ukoliko korisnik izvrši swipe-down otvara se notification drawer, u kome se prikazuje više informacija o notifikaciji
- Iz notification drawer-a korisnik takođe može izvršiti određene akcije nad notifikacijom
- Swipe-down na notifikaciju obično otkriva dodatne akcije

Status bar i notification drawer

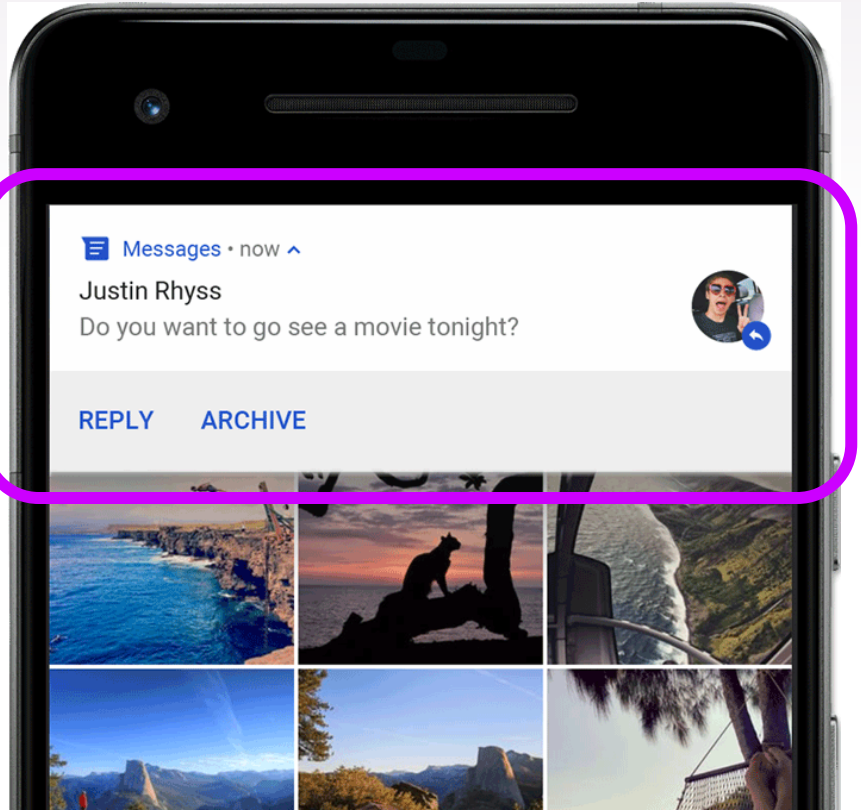


prof. dr Bratislav Predić
dipl. inž. Nevena Tufegdžić

Heads-up notifikacija

- U nekim slučajevima, notifikacija se može prikazati u tzv. floating window formatu preko aktivnosti koja je trenutno pokrenuta
- Ovakvo ponašanje se rezerviše za notifikacije od velike važnosti koje zahtevaju trenutnu akciju korisnika
- Uslov da se ova notifikacija prikaže je da uređaj bude otključan

Heads-up notifikacija



prof. dr Bratislav Predić
dipl. inž. Nevena Tufegdžić

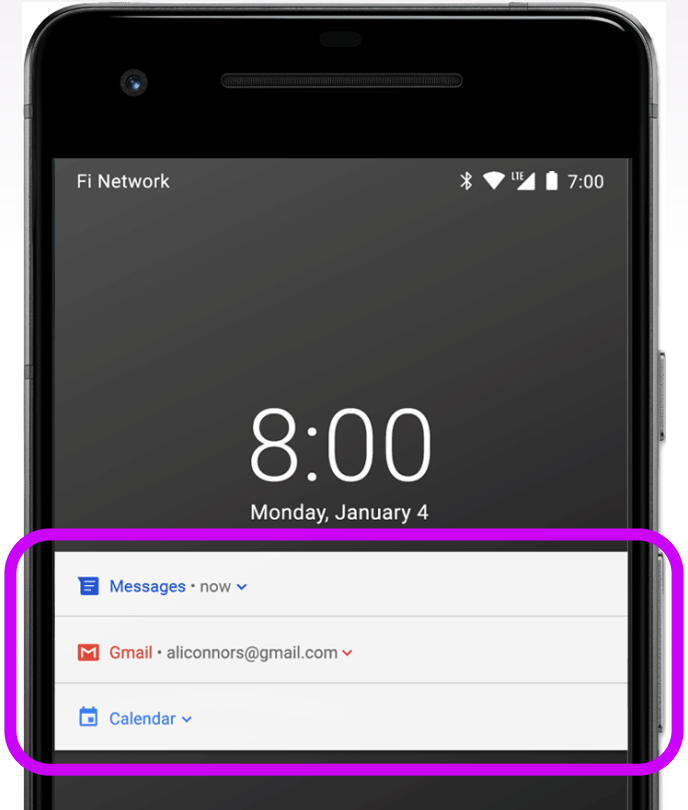
Heads-up notifikacija

- Notifikacija nestaje nakon nekog vremena ali ostaje u notification drawer-u nakon toga
- Slučajevi kada se heads-up notifikacija prikazuje:
  - Korisnička aktivnost je u fullscreen-u
  - Notifikacija ima visoki prioritet i koristi vibracije i zvono (za API 25 i manje)
  - Kanal notifikacije ima visoki prioritet (API 26 i više)

Lock screen notifikacija

- Notifikacije se mogu prikazani na zaključanom ekranu
- Programski se može podesiti nivo detalja koji se prikazuje u okviru notifikacije, kao i uslov da li se notifikacija prikazuje ukoliko uređaj zahteva šifru da bi se otključao
- Korisnici mogu iz sistemskih podešavanja da urede nivo detalja notifikacije, kao i da li se notifikacija uopšte prikazuje na zaključanom ekranu, i to ima prioritet

Lock screen notifikacija



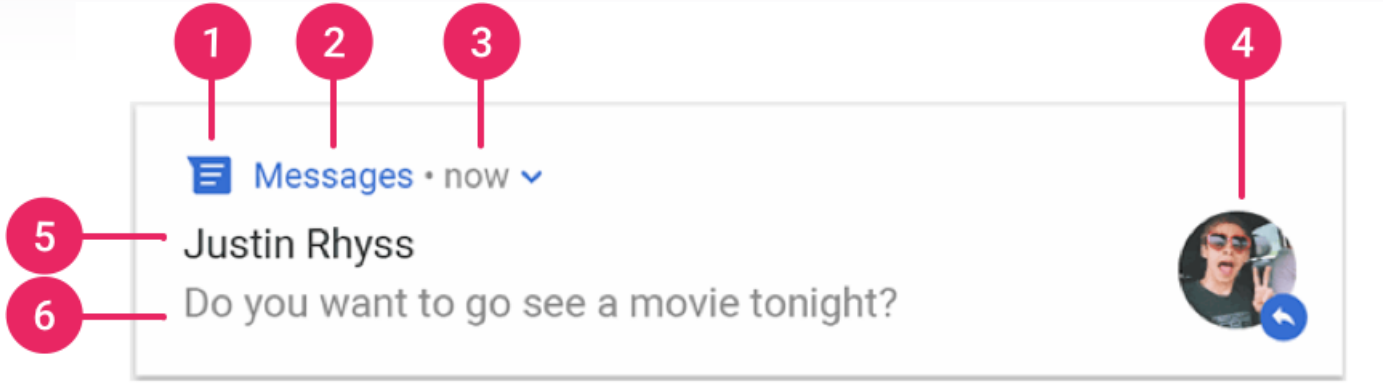
prof. dr Bratislav Predić
dipl. inž. Nevena Tufegdžić

Struktura notifikacije

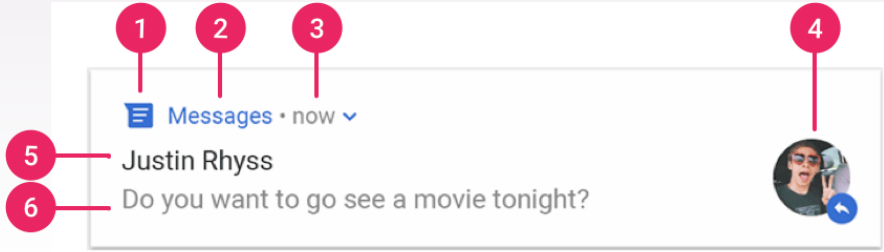
- Dizajn notifikacije zavisi od operativnog sistema, gde on pruža template za izgled notifikacije
- Programer je dužan da popuni delove datog templejta pri kreiranju notifikacije
- Pojedini detalji notifikacije prikazuju se samo u expanded prikazu

Struktura notifikacije

- Najčešći delovi notifikacije:



Struktura notifikacije



1. Small icon – obavezno
2. Ime aplikacije – prosleđuje sistem
3. Timestamp – obezbeđuje sistem, ali može se promeniti programski
4. Large icon – opciono, obično se koristi za profilne slike i slično
5. Naziv notifikacije - opciono
6. Tekst notifikacije - opciono

Kreiranje notifikacije

- Za kreiranje notifikacije koristi se ***NotificationCompat.Builder*** objekat
- Najosnovnija notifikacija sadrži sledeće elemente:
  - Small icon
  - Naslov
  - Tekst
  - Prioritet

Kreiranje notifikacije

```
var builder = NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle(textTitle)
        .setContentText(textContent)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
```

Notification channel

- Kanal za notifikaciju je bitna stavka pri kreiranju notifikacije
- Svaka notifikacija treba da se veže uz neki kanal
- Kanal se prosleđuje preko ID-ja pri kreiranju builder objekta (CHANNEL_ID)
- Pre slanja notifikacije, potrebno je kreirati i registrovati notification channel u sistemu (ukoliko se radi o Android API-ju 26+)

Notification channel

```
private fun createNotificationChannel() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        val name = getString(R.string.channel_name)
        val descriptionText = getString(R.string.channel_description)
        val importance = NotificationManager.IMPORTANCE_DEFAULT
        val channel = NotificationChannel(CHANNEL_ID, name, importance).apply {
            description = descriptionText
        }
        val notificationManager: NotificationManager =
            getSystemService(Context.NOTIFICATION_SERVICE) as
                                                    NotificationManager
        notificationManager.createNotificationChannel(channel)
    }
}
```

Notification channel

- Kreiranje kanala je idempotentna operacija – ukoliko se pozove ista funkcija više puta, ne kreiraju se novi kanali ukoliko jedan već postoji
- Dobro je kreirati kanal čim se aplikacija startuje, budući da nije poznato u kom trenutku korišćenja će kanal biti potreban

Tap action notifikacije

- Svaka notifikacija treba da ima tap akciju
- Ova akcija obično otvara aplikaciju iz koje je stigla notifikacija
- Activity aplikacije otvara se pomoću PendingIntent-a
- Potrebno je povezati notifikaciju sa PendingIntent-om pomoću setContentIntent() metode builder-a

Tap action notifikacije

```
val intent = Intent(this, AlertDetails::class.java).apply {
    flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
}
val pendingIntent: PendingIntent = PendingIntent.getActivity(this, 0, intent,
        PendingIntent.FLAG_IMMUTABLE)

val builder = NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!")
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .setContentIntent(pendingIntent)
        .setAutoCancel(true)
```

Prikaz notifikacije

- Prikaz notifikacije vrši se pozivanjem metode NotificationManager-a: *notify()*
- Ova metoda prihvata ID notifikacije i notifikaciju dobijenu iz builder-a
- ID treba da bude jedinstveni integer koji definiše programer

```
with(NotificationManagerCompat.from(this)) {
    notify(notificationId, builder.build())
}
```

Uvod u Firebase

prof. dr Bratislav Predić
dipl. inž. Nevena Tufegdžić

Navigacija u Androidu
Razvoj mobilnih aplikacija i servisa

Firebase

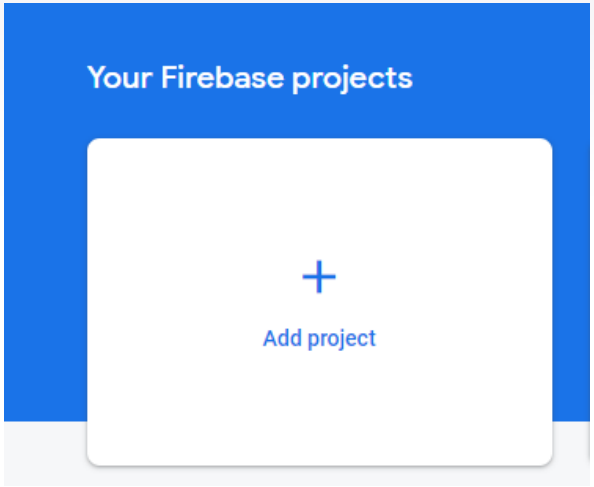
- Platforma za razvoj aplikacija korišćenjem cloud servisa
- Koristi se kao backend servis pri razvoju mobilnih i Web aplikacija, kao i igara
- Nudi veliki broj servisa

Firebase servisi

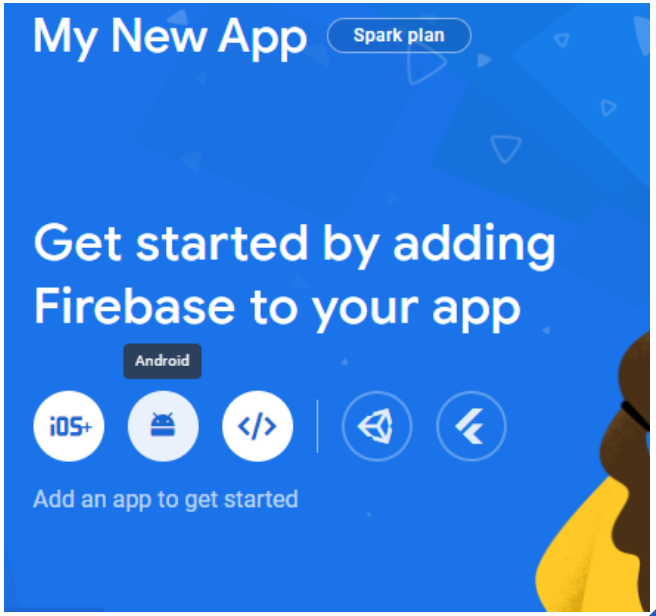
1. **Authentication**
2. **NoSQL baze podataka**
3. **Storage hosting servis**
4. Release & Monitor servisi
5. User Engagement servisi

Dodavanje Firebase-a u aplikaciju

1. Kreirati Firebase projekat na Firebase Console



2. Registrovati aplikaciju na Firebase-u i pratiti korake na konzoli



Dodavanje Firebase-a u aplikaciju

- U app gradle fajl potrebno je dodati sledeće:

```
implementation 'com.google.firebase:firebase-analytics'
implementation platform('com.google.firebase:firebase-bom:31.1.0')

implementation 'com.google.firebase:firebase-analytics-ktx'
implementation 'com.google.firebase:firebase-auth-ktx'
implementation 'com.google.firebase:firebase-firestore-ktx'
implementation 'com.google.firebase:firebase-storage-ktx'
implementation 'com.google.firebase:firebase-messaging-ktx'
implementation 'com.google.firebase:firebase-database-ktx'
```

Authentication

- Predstavlja preporučeni način za autentikaciju korisnika u *Firebase*-u
- Podržava kreiranje korisničkih naloga koristeći gotove funkcije
- Pribavljanje Authentication objekta:

```
private var auth: FirebaseAuth = Firebase.auth
```

Authentication

- *Sign-in* methods:
  - *E-mail & password*
  - *Phone*
  - Nalozi na ostalim platformama (*Google*, *Facebook*, *Twitter*, itd.)
- *Auth* se dodaje u projekat dodavanjem odgovarajućeg *dependency*-ja u *app-level Gradle* fajl

Authentication

- Primer kreiranja naloga pomoću email-a i šifre:

```
fun createAccount(email: String, password: String){
    auth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener() { task ->
            if (task.isSuccessful) {
                // kreiran nalog
            } else {
                // obrada greške
            }
    }
}
```

Authentication

- Primer logovanja pomoću email-a i šifre:

```
fun login(email: String, password: String){
    auth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener() { task ->
            if (task.isSuccessful) {
                // korisnik je ulogovan
            } else {
                // obrada greške
            }
    }
}
```

UID

- UID predstavlja jedinstveni identifikator koji *Firebase Auth* automatski dodeljuje svakom korisniku pri registraciji
- Korišćenjem UID-ja može se referencirati svaki korisnik aplikacije
- Dobra praksa je da se informacije o korisniku u *NoSQL* bazama čuvaju na osnovu korisničkog UID-ja

User Profile

- *User Profile* je deo *Auth*-a u kome se čuvaju osnovne informacije o trenutno ulogovanom korisniku (URL profilne slike, UID, *e-mail*)
- Pristup trenutno logovanom korisniku se vrši pomoću sledećeg koda: `Firebase.auth.currentUser`

User Profile

- Moguće je ažurirati profil i promeniti šifru preko currentUser-a
  - Auth.currentUser.updatePassword()
  - Auth.currentUser.updateProfile()

Promena informacija u profilu

```
val profileUpdate = userProfileChangeRequest {
    displayName = "New Name"
    photoUri = Uri.parse(imageUrl)
}
auth.currentUser!!.updateProfile(profileUpdate)
```

Promena šifre

```
fun changePassword(email: String){
    auth.sendPasswordResetEmail(email.value!!).addOnCompleteListener {
        if (it.isSuccessful) {
            // uspešno poslat email za resetovanje šifre
        } else {
            // obrada greške
        }
    }
}

auth.currentUser!!.updatePassword(password)
```

Realtime Database

- Predstavlja NoSQL bazu fokusiranu na praćenje brzih promena u podacima
- Pri svakoj promeni podataka u bazi, svaki uređaj koji prati te podatke dobija ažuriranu verziju u roku od par milisekundi
- RTDB je pogodna za aplikacije u kojima se podaci često menjaju, kao što je chat aplikacija

Operacije u RTDB

- RTDB podržava sve CRUD operacije
- Praćenje podataka u bazi se vrši korišćenjem event listener-a
- Pre rada sa podacima potrebno je pribaviti referencu na bazu:

```
val database = Firebase.database(dbUrl)
```

Upis u RTDB

- RTDB radi sa referencama – enkapsulirane u klasi DatabaseReference
- Svi podaci u bazi struktuirani su u obliku JSON stabla

```
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      "contacts": { "ghopper": true },
    },
    "ghopper": { ... },
    "eclarke": { ... }
  }
}
```

Upis u RTDB

```
fun writeNewUser(userId: String, name: String, email: String) {
    val user = User(name, email)
    database.child("users").child(userId).setValue(user)
}
```

- User klasa je data klasa – potrebno je da se tako implementira kako bi mogla da se serijalizuje u JSON objekat

```
data class User(val username: String? = null,
        val email: String? = null) { }
```

Čitanje iz RTDB

- Read operacija se može obaviti korišćenjem get() metode nad referencom objekta u bazi:

```
mDatabase.child("users").child(userId).get().addOnSuccessListener {
    Log.i("firebase", "Got value ${it.value}")
}.addOnFailureListener{
    Log.e("firebase", "Error getting data", it)
}
```

- Ova metoda čita samo trenutnu vrednost objekta i to tačno jednom

Čitanje iz RTDB

- Ukoliko je potrebno da se operacija čitanja obavlja svaki put kada dođe do promene podataka u bazi, koristi se ValueEventListener
- Ovo je objekat koji u sebi sadrži dve bitne metode:
  - onDataChange() – poziva se pri promeni podataka na referenci
  - onCancelled() – pozvaće se u slučaju neke greške
- Potrebno je override-ovati ove metode

Čitanje iz RTDB

```
val userListener = object : ValueEventListener {
    override fun onDataChange(dataSnapshot: DataSnapshot) {
        // Preuzima se User objekat i vrši se neka obrada
        val post = dataSnapshot.getValue<Post>()
    }

    override fun onCancelled(databaseError: DatabaseError) {
        // Obrada greške
        Log.w(TAG, "loadUser:onCancelled", databaseError.toException())
    }
}
userReference.addValueEventListener(userListener)
```

Brisanje iz RTDB

- Operacija brisanja iz baze vrši se pozivom metode removeValue() na referencu iz RTDB:

```
mDatabase.child("users").child(userId).removeValue()
```

- Alternativno, može se iskoristiti metoda setValue() kojoj se prosleđuje null:

```
mDatabase.child("users").child(userId).setValue(null)
```

Firestore

- Predstavlja fleksibilnu i skalabilnu *NoSQL* bazu
- Podaci su organizovani u kolekcije i dokumente
- Podržava transakcionalnost za *write* operacije
- Za razliku od RTDB, podržava operacije za nizovima
- Poseduje automatski *scaling*

Operacije sa Firestore

- Podržava sve CRUD operacije
- Podržava kompleksnije upite
- Pre rada sa podacima potrebno je pribaviti referencu na bazu:

```
private val db = Firebase.firestore
```

Čitanje iz Firestore

- Baza je organizovana po dokumentima, tako da se read operacija obavlja na sledeći način:

```
db.collection("places").document(placeID).get().addOnSuccessListener {
    selectedPlace = it.toObject<Place>()
}
```

Upis u Firestore

- Upis i ažuriranje podataka vrši se set() metodom:

```
db.collection("places").document(placeID)
    .set(place)
    .addOnSuccessListener {
        Log.d(TAG, "DocumentSnapshot successfully written!")
    }
    .addOnFailureListener {
        e -> Log.w(TAG, "Error writing document", e)
    }
```

Brisanje iz Firestore

- Brisanje podataka vrši se delete() metodom:

```
db.collection("places").document(placeID)
    .delete(place)
    .addOnSuccessListener {
        Log.d(TAG, "DocumentSnapshot successfully deleted!")
    }
    .addOnFailureListener {
        e -> Log.w(TAG, "Error deleting document", e)
    }
```

Firestore Queries

- Primer jednog jednostavnog upita:

```
query = db.collection("places").orderBy("name")
```

- Primer kompleksnijeg upita koji ispituje da li se neki element nalazi u nizu koji je deo dokumenta:

```
query = db.collection("places").orderBy("name")
    .whereArrayContains("nameQueryList", placeNameFilter)
```

Cloud Storage

- Predstavlja servis za skladištenje podataka koje korisnici generišu u okviru korišćenja aplikacije
- Obično služi za skladištenje slika, videa i drugih vrsta fajlova
- Skalabilan i efikasan

```
private fun uploadPicture(uuid: String): Task<Uri> {
    val imageRef: StorageReference = storage.reference.child( pathString: "places")
        .child(uuid).child( pathString: "${uuid}.jpg")
    val baos = ByteArrayOutputStream()
    val bitmap = picture.value
    bitmap!!.compress(Bitmap.CompressFormat.JPEG,  quality: 100, baos)
    val data = baos.toByteArray()
    val uploadTask = imageRef.putBytes(data)
    val urlTask = uploadTask.continueWithTask{ task ->
        if (!task.isSuccessful) {
            task.exception?.let {  it: Exception
                _actionState.value = ActionState.ActionError("Upload error: ${it.message}")
            }
        }
        imageRef.downloadUrl  ^continueWithTask
    }
    return urlTask
}
```

Upload-ovanje slike

Dobijanje URL-a upload-ovane slike

```
uploadPicture(uuid).addOnCompleteListener{task ->
    if(task.isSuccessful){
        val imageUrl = task.result.toString()
        putPlace(uuid, imageUrl)
    }
}
```

Preuzimanje slike iz Storage-a

```
Glide.with(context!!).load(viewModel.selectedPlace!!.pictureUrl)
    .skipMemoryCache( skip: true).diskCacheStrategy(
    DiskCacheStrategy.NONE).into(binding.placePictureDetails)
```

Literatura

- Notifications with Jetpack Compose
- Notifications - Android Docs
- Firebase dokumentacija
- Firebase - Android Codelab

Hvala na pažnji!