MOBILNE APLIKACIJE

Vežbe 9

SQLite

2022/2023

Sadržaj

1. SQLite

Android aplikacije mogu da koriste ugradjen sistem za upravljanje bazama podataka (*SQLite*). *SQLite* se izvršava u istom procesu kao i sama aplikacija.

Baza podataka predstavljena je klasom *SQLiteDatabase.*

CRUD operacije nad bazom podataka izvršavaju se pozivom *insert, query, update i delete* metoda.

query metoda je specifična jer kao rezultat vraća *Cursor*. *Cursor* vraća adresu gde se podaci nalaze tj. relacija koja je rezultat SQL upita predstavljena je kursorom. Kursore koristimo da bismo vršili navigaciju kroz rezultat upita.

U bazi svaka tabela ima redove i kolone. Potrebno je pozicionirati se na odgovarajući red i odgovarajuću kolonu. Prvo se pozicioniramo na red sa nekom od metoda:

- `boolean move(int offset)`

  Pomeri se za određeni broj redova od trenutne pozicije.
- `boolean moveToFirst()`

  Pređi na prvi red.
- `boolean moveToLast()`

  Pređi na poslednji red
- `boolean moveToNext()`

  Pređi na sledeći red.
- `boolean moveToPrevious()`

  Pređi na prethodni red.

Kada se pozicioniramo na red, sledeći korak je da odaberemo kolonu, tj. da pročitamo rezultat upita. Za to možemo da iskoristimo neku od metoda:

- `int getCount()`
- `int getColumnIndex(String column_name)`
- `String getColumnName(int column_index)`
- `String getString(int column_index)`
- `int getInt(int column_index)`
- `long getLong(int column_index)`
- `float getFloat(int column_index)`

U primeru za ove vežbe kreirali smo klasu *ReviewerSQLiteHelper* koja nasleđuje klasu *SQLiteOpenHelper*. Klasa *SQLiteOpenHelper* nam omogućava da napravimo, izmenimo ili otvorimo bazu podataka.

Implementiramo metode:

- `void onCreate(SQLiteDatabase database)`

- *void onOpen(SQLiteDatabase database)*
- *void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion)*
- *void onDowngrade(SQLiteDatabase database, int oldVersion, int newVersion)*

onCreate

U ovoj metodi kreiramo našu bazu tako što pozivamo *db.execSQL* za kreiranu tabelu (slika 1).

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL(DB_CREATE);
}
```

Slika 1. Kreiranje tabele

onUpgrade

Metodu *onUpgrade* pozivamo kad baza treba da se promeni (slika 2). Prvi korak je da dropujemo sve tabele koje imamo.

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_CINEMA);
    onCreate(db);
}
```

Slika 2. Izmena tabele

U klasi *Util* kreirali smo metodu *initDB* koja služi za inicijalizaciju naše baze (slika 3). U toj metodi smo kreirali dva entiteta i uneli ih u tabelu pozivanjem metode *insert*.

```java
public class Util {
    public static void initDB(Activity activity) {
        ReviewerSQLiteHelper dbHelper = new ReviewerSQLiteHelper(activity);
        SQLiteDatabase db = dbHelper.getWritableDatabase();
        {
            ContentValues entry = new ContentValues();
            entry.put(ReviewerSQLiteHelper.COLUMN_NAME, "Arena");
            entry.put(ReviewerSQLiteHelper.COLUMN_DESCRIPTION, "Cineplexx 3D");
            entry.put(ReviewerSQLiteHelper.COLUMN_AVATAR, -1);

            activity.getContentResolver().insert(DBContentProvider.CONTENT_URI_CINEMA, entry);

            entry = new ContentValues();
            entry.put(ReviewerSQLiteHelper.COLUMN_NAME, "Cinestar");
            entry.put(ReviewerSQLiteHelper.COLUMN_DESCRIPTION, "Najnoviji 5D");
            entry.put(ReviewerSQLiteHelper.COLUMN_AVATAR, -1);

            activity.getContentResolver().insert(DBContentProvider.CONTENT_URI_CINEMA, entry);
        }

        db.close();
    }
```

Slika 3. Metoda za inicijalizacije *SQLite* baze

Na slici 4 se vidi da se metoda *initDB,* klase *Util,* poziva u trenutku kada korisnik klikne na stavku menija *refresh.* Kada pokrenete aplikaciju na uređaju videćete da se, nakon što kliknete na dugme *refresh,* pojave dva filma, koja smo kreirali u metodi *initDB*.

```java
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_settings:
                Intent i = new Intent( packageContext: this, ReviewerPreferenceActivity.class);
                startActivity(i);
                return true;
            case R.id.action_new:
                Util.initDB( activity: MainActivity.this);
                finish();
                startActivity(getIntent());
        }

        return super.onOptionsItemSelected(item);
    }
```

Slika 4. Pozivanje metode initDB

U klasi *Util* kreirali smo još metode *insert, update, delete i query* koje služe za obavljanje istoimenih funkcija ka bazi. Na slici 6 se vidi da se metode, klase *Util,* pozivaju u trenutku kada korisnik klikne na stavku menija (ostali case slucajevi).

```java
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_settings:
            Intent i = new Intent( packageContext: this, ReviewerPreferenceActivity.class);
            startActivity(i);
            return true;
        case R.id.action_refresh:
            Util.initDB( activity: MainActivity.this);
            finish();
            startActivity(getIntent());
            return true;
        case R.id.action_insert:
            Util.insert( activity: MainActivity.this);
            finish();
            startActivity(getIntent());
            return true;
        case R.id.action_update:
            Util.update( activity: MainActivity.this);
            finish();
            startActivity(getIntent());
            return true;
        case R.id.action_delete:
            Util.delete( activity: MainActivity.this);
            finish();
            startActivity(getIntent());
            return true;
    }
```

Slika 6. Pozivanje metoda insert, update i delete

2. Dobavljači sadržaja

Dobavljači sadržaja (*ContentProvider*) upravljaju podacima aplikacije i omogućavaju da im se pristupi na standardizovan način. Podaci se mogu nalaziti u bazi podataka, na internetu ili nekom drugom mestu. Npr. postoji dobavljač sadržaja koji dozvoljava korisniku da uređuje listu kontakata na uređaju. Ovo znači da svaka aplikacija, ako ima odobrenje, može da pristupa određenim dobavljačima i da čita ili upisuje neke podatke.
Postoje dve vrste dobavljača sadržaja:

1. Sistemski
2. Aplikacioni

Sistemski dobavljači su uključeni u Android (*Browser, Calendar, CallLog, Contacts*..).

Aplikacione dobavljače sadržaja prave programeri, koji su zaduženi za kreiranje same aplikacije. Ako želimo da resursima naše aplikacije mogu da pristupe neke druge aplikacije, onda moramo to i da im omogućimo kreiranjem dobavljača sadržaja. Kako se prave aplikacioni dobavljači sadržaja predstavljeno je u potpoglavlju 2.2.

2.1 Jedinstvena identifikacija resursa

Resurse opisuje URI i MIME tip.

Primer URI-ja:

content://user_dictionary/words

URI se sastoji iz 3 dela:

- šeme,
- imena dobavljača i
- imena tabele.

MIME tip specifira tip sadržaja (text/plain, text/pdf, image/jpeg, itd.). Kada se navede kao tip npr. *jpeg* to znači da ćemo dobiti samo slike sa tom ekstenzijom.

Primer: U galeriji telefona se čuvaju slike koje kamera napravi. Galeriji pristupamo preko dobavljača sadržaja i tražimo određenu sliku. Kao povratna vrednost stiže URI, a ne slika, jer takva situacija ne bi bila optimalna. Sa tim URI-jem možemo da tražimo direktan pristup resursu.

2.2 Aplikacioni dobavljači sadržaja

Kreiramo klasu *DBContentProvider* koja nasleđuje *ContentProvider.*

Deklarišemo klasu *DBContentProvider* u *AndroidManifest.xml* datoteci (slika 7) tako što navedemo element *<provider>.*

```
<provider
    android:name="rs.reviewer.database.DBContentProvider"
    android:authorities="rs.reviewer"
    android:exported="false" />
```

Slika 7. Deklarisanje dobavljača sadržaja u *AndroidManifest* datoteci

Na slici 8 se nalazi primer kreiranja URI-ja. *Authority* polje je ime dobavljača, što je najčešće paket. *Cinema_path* je naziv tabele.

Ako korisnik želi informacije o pojedinačnom resursu onda definišemo *UriMatcher.* Svaki put proveravamo da li je zahtev stigao za celu tabelu ili za pojedinačan red i u zavisnosti od toga se pretražuje baza i vraća se rezultat nazad.

MIME tip specificira tip sadržaja i kada želimo da dobijemo sve slike sa ekstenzijom jpeg, onda navodimo image/jpeg, a ako želimo sve slike nezavisno od ekstenzije, šaljemo image/*.

```
private static final int CINEMA = 10;
private static final int CINEMA_ID = 20;

private static final String AUTHORITY = "rs.reviewer";

private static final String CINEMA_PATH = "cinema";

public static final Uri CONTENT_URI_CINEMA = Uri.parse("content://" + AUTHORITY + "/" + CINEMA_PATH);

private static final UriMatcher sURIMatcher = new UriMatcher(UriMatcher.NO_MATCH);

static {
    sURIMatcher.addURI(AUTHORITY, CINEMA_PATH, CINEMA);
    sURIMatcher.addURI(AUTHORITY,  path: CINEMA_PATH + "/#", CINEMA_ID);
}
```

Slika 8. Kreiranje URI-ja

Prilikom nasleđivanja *ContentProvider* klase, treba implementirati metode:

- *insert()*,
- *query()*,
- *update()*,
- *delete()*,
- *getType()* i
- *onCreate()*

Na slici 9 se nalazi primer metode *insert*.

```java
@Nullable
@Override
public Uri insert(Uri uri, ContentValues values) {
    Uri retVal = null;
    int uriType = sURIMatcher.match(uri);
    SQLiteDatabase sqlDB = database.getWritableDatabase();
    long id = 0;
    switch (uriType) {
        case CINEMA:
            id = sqlDB.insert(ReviewerSQLiteHelper.TABLE_CINEMA, nullColumnHack: null, values);
            retVal = Uri.parse(CINEMA_PATH + "/" + id);
            break;
        default:
            throw new IllegalArgumentException("Unknown URI: " + uri);
    }
    getContext().getContentResolver().notifyChange(uri, observer: null);
    return retVal;
}
```

Slika 9. Metoda *insert*

Na slici 10 se nalazi primer *query* metode, koja kao rezultat vraća *Cursor.*

```java
@Nullable
@Override
public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {
    // Uisng SQLiteQueryBuilder instead of query() method
    SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();

    // check if the caller has requested a column which does not exist //checkColumns(projection);
    int uriType = sURIMatcher.match(uri);
    switch (uriType) {
        case CINEMA_ID:
            // Adding the ID to the original query
            queryBuilder.appendWhere( inWhere: ReviewerSQLiteHelper.COLUMN_ID + "="
                    + uri.getLastPathSegment());
            //$FALL-THROUGH$
        case CINEMA:
            // Set the table
            queryBuilder.setTables(ReviewerSQLiteHelper.TABLE_CINEMA);
            break;
        default:
            throw new IllegalArgumentException("Unknown URI: " + uri);
    }

    SQLiteDatabase db = database.getWritableDatabase();
    Cursor cursor = queryBuilder.query(db, projection, selection,
            selectionArgs, groupBy: null, having: null, sortOrder);
    // make sure that potential listeners are getting notified
    cursor.setNotificationUri(getContext().getContentResolver(), uri);
    return cursor;
```

Slika 10. Metoda *query*

U primeru za ove vežbe možete pogledati i sve ostale metode.

Šta ako se pojave dve aplikacije koje u isto vreme pošalju zahtev za neki resurs naše aplikacije? Treba da vodimo računa o sinhronizaciji. Sve metode osim metode *onCreate()* treba da budu *thread-safe*. Treba izbegavati izvršavanje dugačkih operacija u metodi *onCreate()*.

Primećujemo da metoda *onDestroy* ne postoji, jer dobavljači sadržaja postoje od početka do kraja procesa.

Klasa *ContentResolver* omogućava izvršavanje CRUD (create, read, update, delete) operacija nad skladištem podataka. Podacima pristupamo tako što zatražimo odgovarajuća prava pristupa, a potom izvršavamo upit nad dobavljenim sadržajem.

Upit se postavlja na sličan način na koji se postavlja SQL upit. Sadrži URI, spisak kolona koje treba vratiti (projekciju), uslov koji vraćene vrste treba da zadovolje (selekciju) i način sortiranja rezultata.

Na sličan nacin na koji je moguće pristupiti podacima, moguće ih je i promeniti.

Primer upotrebe *ContentResolver*-a možete videti na slici 3 kada se vrši inicijalizacija SQLite baze. Metodi *initDB* prosleđujemo aktivnost, u našem primeru se prosleđuje *MainActivity*, i pozivamo metodu *getContentResolver*, a potom i metodu *insert* da ubacimo kreirani film.

3. Punjači

Punjači (*Loader*) omogućavaju asinhrono učitavanje podataka u aktivnosti i fragmente. Oni nadgledaju izvore podataka i isporučuju sadržaj kada se podaci promene. Takođe, oni vode računa o promeni stanja aktivnosti ili fragmenta.

SimpleCursorAdapter povezuje kursor sa *ListView* ili *GridView* pogledom. Na slici 11 se nalazi primer upotrebe *SimpleCursorAdapter*-a.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Toast.makeText(getActivity(), text: " My Fragment - onCreate()", Toast.LENGTH_SHORT).show();
    /*...*/
    LoaderManager.getInstance(this).initLoader( id: 0,  args: null,  callback: this);

    String[] from = new String[] { ReviewerSQLiteHelper.COLUMN_NAME, ReviewerSQLiteHelper.COLUMN_DESCRIPTION };
    int[] to = new int[] {R.id.name, R.id.description};
    adapter = new SimpleCursorAdapter(getActivity(), R.layout.cinema_list,  c: null, from, to,  flags: 0);
    setListAdapter(adapter);

}
```

Slika 11. Primer upotrebe *SimpleCursorAdapter*-a

4. Firebase

Firebase je platforma za razvoj mobilnih i web aplikacija koju je razvio Google. Ova platforma nudi razne alate i usluge koji omogućavaju programerima da brzo i jednostavno razvijaju i skaliraju svoje aplikacije. Firebase podržava različite vrste aplikacija, uključujući Android, iOS i web aplikacije.

Neke od glavnih usluga koje Firebase nudi su:

- Realtime Database: Firebase baza podataka u stvarnom vremenu koja omogućuje aplikacijama da sinhronizuju podatke između korisnika u stvarnom vremenu.
- Cloud Firestore: moderna baza podataka koja se temelji na dokumentima, a koristi se za pohranu i sinhronizaciju podataka između korisnika i aplikacija.
- Authentication: Omogućava korisnicima da se prijave u aplikaciju putem e-pošte, Facebook-a, Google-a i drugih usluga.
- Storage: Firebase omogućuje spremanje datoteka u oblaku, što je korisno za aplikacije koje trebaju spremiti slike, videozapise i druge vrste medija.
- Cloud Messaging: Firebase pruža uslugu push obavijesti koja omogućuje aplikacijama da šalju obavijesti na mobilne uređaje.
- Hosting: Firebase nudi brzo i jednostavno hosting za web aplikacije.

Firebase također nudi i druge usluge poput analitike, testiranja i optimizacije aplikacija, dinamičkih veza, upravljanja korisnicima i još mnogo toga.

4.1. Realtime Database

Realtime Database je NoSQL baza podataka koja je specijalizovana za sinhronizaciju podataka u stvarnom vremenu između korisnika i aplikacija.

Realtime Database koristi JSON format za pohranu i sinhronizaciju podataka. Podaci se pohranjuju u obliku stabla JSON objekata, što omogućuje brzo i jednostavno upravljanje podacima. Realtime Database podržava sinhronizaciju podataka u stvarnom vremenu između korisnika i aplikacija, što znači da se promene u podacima automatski ažuriraju na svim uređajima koji koriste aplikaciju.

Neki od glavnih usluga koje nudi Realtime Database su:

- Realtime sinhronizacija: Realtime Database omogućuje automatsku sinhronizaciju podataka u stvarnom vremenu između korisnika i aplikacija. To znači da se promene u podacima automatski ažuriraju na svim uređajima koji koriste aplikaciju.
- Realtime događaji: Realtime Database omogućuje praćenje događaja u stvarnom vremenu, poput dodavanja, brisanja i ažuriranja podataka. To je korisno za aplikacije koje zahtevaju praćenje promena u podacima.
- Offline podrška: Realtime Database omogućuje rad s podacima i kada nema internetske veze. To je korisno za korisnike koji se nalaze u područjima s lošim pristupom internetu.

- Sigurnosna pravila: Realtime Database omogućuje postavljanje pravila sigurnosti za pristup podacima, što pomaže u zaštiti podataka od neovlaštenog pristupa.
- Integracija s drugim Firebase uslugama: Realtime Database se lako integrira s drugim Firebase uslugama kao što su autentifikacija korisnika, upravljanje korisnicima i analitika.

4.2. Cloud Firestore

Cloud Firestore je moderna baza podataka koja se temelji na dokumentima, a koristi se za pohranu i sinhronizaciju podataka između korisnika i aplikacija.

Firestore se temelji na NoSQL modelu podataka, što znači da podaci nisu pohranjeni u relacionim tabelama kao u tradicionalnoj relacionoj bazi podataka. Umesto toga, podaci se pohranjuju u kolekcijama koje sadrže dokumente, a dokumenti mogu sadržavati različite vrste podataka, uključujući tekst, brojeve, datume, objekte i nizove.

Firestore se odlikuje brzim i skalabilnim pristupom podacima, a koristi se za različite vrste aplikacija, uključujući mobilne i web aplikacije, igre i druge usluge.

Firestore ima i neke druge opcije, uključujući:

- Realtime komunikacija: Firestore omogućuje izvođenje upita u stvarnom vremenu, što znači da se podaci automatski ažuriraju kad se promene.
- Indexiranje polja: Firestore automatski indeksira polja u dokumentima, što omogućuje brzo pretraživanje podataka i postavljanje upita.
- Sigurnosna pravila: Firestore omogućuje programerima postavljanje pravila sigurnosti za pristup podacima, što pomaže u zaštiti podataka od neovlaštenog pristupa.
- Integracija s drugim Firebase uslugama: Firestore se lako integriše s drugim Firebase uslugama kao što su autentifikacija korisnika, upravljanje korisnicima i analitika.

4.3.1. Implementacija

Da biste koristili Cloud Firestore u vašoj Java aplikaciji, prvo trebate uključiti Firebase Firestore biblioteku u vaš projekt (slika 12). Postupak za uključivanje biblioteke zavisi od vašeg razvojnog okruženja, ali obično se radi o dodavanju biblioteke u datoteku ***build.gradle*** vašeg projekta.

```
dependencies {

    implementation 'androidx.appcompat:appcompat:1.5.1'
    implementation 'com.google.android.material:material:1.7.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    implementation 'com.google.firebase:firebase-firestore:24.6.0'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'

}
```

Slika 12. Firebase Firestore biblioteka

Nakon uključivanja biblioteke, možete koristiti Firestore klijent za pristup i manipulisanje podacima u vašoj Firestore bazi podataka.

Selektovanje, dodavanje, izmena i brisanje podataka u Firestore bazi podataka implementirane su u *CloudStoreUtil* klasi.

Dodavanje podataka

Dodavanje novog objekta u Firestore bazu podataka dato je na slici 13.

```
public static void insert(){
    // kreiraj novi objekat klase User
    User user1 = new User("Milica", "Milic");
//      Map<String, Object> user1 = new HashMap<>();
//      user.put("firstName", "Ivana");
//      user.put("lastName", "Ivic");

    FirebaseFirestore db = FirebaseFirestore.getInstance();
//      dodaje se novi user u kolekciju "users"
    db.collection( collectionPath: "users") CollectionReference
            .add(user1) Task<DocumentReference>
            .addOnSuccessListener(new OnSuccessListener<DocumentReference>() {
                @Override
                public void onSuccess(DocumentReference documentReference) {
                    Log.d( tag: "REZ_DB",  msg: "DocumentSnapshot added with ID: " + documentReference.getId());
                }
            })
            .addOnFailureListener(new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
                    Log.w( tag: "REZ_DB",  msg: "Error adding document", e);
                }
            });
```

Slika 13. Dodavanje novog user-a

U ovom primeru, stvaramo novi objekat podataka pomoću klase HashMap, a zatim ga dodajemo u kolekciju "users" pomoću Firestore klijenta. Metoda ***addOnSuccessListener*** se poziva kada je user uspešno dodat u bazu podataka, a ***addOnFailureListener*** se poziva ako se dogodi greška prilikom dodavanja user-a.

Dobavljanje podataka

Dobavljanje sadržaja neke tabele iz Firestore baze podataka dato je na slici 14.

```
public static void select(){
    FirebaseFirestore db = FirebaseFirestore.getInstance();
    db.collection( collectionPath: "users") CollectionReference
            .get() Task<QuerySnapshot>
            .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                @Override
                public void onComplete(@NonNull Task<QuerySnapshot> task) {
                    if (task.isSuccessful()) {
                        for (QueryDocumentSnapshot document : task.getResult()) {
                            Log.d( tag: "REZ_DB",  msg: document.getId() + " => " + document.getData());
                        }
                    } else {
                        Log.w( tag: "REZ_DB",  msg: "Error getting documents.", task.getException());
                    }
                }
            });
}
```

Slika 14. Dobavljanje tabele users

U ovom primeru koristimo metodu ***get()*** Firestore klijenta kako bismo dobili sve dokumente iz kolekcije "users". Metoda ***addOnCompleteListener*** se poziva kada se dobave podaci, a zatim se prolazi kroz sve dokumente korištenjem petlje i ispisuje se ID dokumenta i podaci.

Izmena podataka

Izmena sadržaja nekog objekta iz Firestore baze podataka dato je na slici 15.

```
public static void update(){
    FirebaseFirestore db = FirebaseFirestore.getInstance();
    // izmena dokumenta s ID-em "GFowOuJW63DykTxughJS" iz kolekcije "users"
    DocumentReference docRef = db.collection( collectionPath: "users").document( documentPath: "GFowOuJW63DykTxughJS");
    docRef
            .update( field: "firstName",  value: "Dragan")
            .addOnSuccessListener(aVoid -> Log.d( tag: "REZ_DB",  msg: "User successfully changed"))
            .addOnFailureListener(e -> Log.w( tag: "REZ_DB",  msg: "Error getting documents.", e));

}
```

Slika 15. Izmena objekta iz Firestore baze

U ovom primeru koristimo metodu ***update()*** Firestore klijenta kako bismo izmenili podatak "firstName" dokumenta s ID-em "GFowOuJW63DykTxughJS" iz kolekcije "users". Metoda ***addOnSuccessListener*** se poziva kada je dokument uspešno izmenjen, a metoda ***addOnFailureListener*** se poziva ako se dogodi greška.

Brisanje podataka

Brisanje nekog objekta iz Firestore baze podataka dato je na slici 16.

```
public static void delete(){
    FirebaseFirestore db = FirebaseFirestore.getInstance();
    // izbrisati user-a s ID-om "1wUqKBOWBI5O1Iq6rOAA" iz kolekcije "users"
    db.collection( collectionPath: "users") CollectionReference
            .document( documentPath: "1wUqKBOWBI5O1Iq6rOAA") DocumentReference
            .delete() Task<Void>
            .addOnSuccessListener(aVoid -> Log.d( tag: "REZ_DB",  msg: "The user has been deleted."))
            .addOnFailureListener(e -> Log.w( tag: "REZ_DB",  msg: "Error deleting document.", e));

}
```

Slika 16. Brisanje objekta iz Firestore baze

Ovaj kod koristi metodu ***delete*** Firestore klijenta za brisanje dokumenta s ID-om "1wUqKBOWBI5O1Iq6rOAA" iz kolekcije "users". Metoda ***addOnSuccessListener*** i ***addOnFailureListener*** se pozivaju kada se brisanje dokumenta završi uspešno ili neuspešno, te ispisuju odgovarajuće poruke u logu.

4.4. Firebase Assistant

Firebase Assistant je alat koji se integriše u Android Studio i omogućava jednostavno dodavanje Firebase funkcionalnosti u aplikaciju. Firebase Assistant pruža vizuelni prikaz Firebase usluga i omogućava brzo dodavanje usluga u aplikaciju.

Kada se Firebase Assistant otvori u Android Studiju (Tools/Firebase opcija), prikazuju se različite usluge koje Firebase nudi, kao što su Realtime Database, Cloud Firestore, autentifikacija korisnika i analitika. Korisnici mogu jednostavno odabrati usluge koje žele koristiti i slediti uputstva za konfiguraciju.

Firebase Assistant također omogućava pristup dokumentaciji i primerima koda za svaku Firebase uslugu, što je korisno za početnike koji se tek upoznaju s Firebase platformom.

Korištenje Firebase Assistanta u Android Studio-u olakšava dodavanje Firebase funkcionalnosti u aplikaciju, ubrzava razvoj i pomaže programerima u stvaranju boljih aplikacija.

5. Domaći

Domaći se nalazi na *Canvas-u (canvas.ftn.uns.ac.rs)* na putanji *Vežbe/09 Zadatak.pdf*

Primer *Vezbe9_1 i Vezbe9_2* možete preuzeti na sledećem linku:
https://gitlab.com/antesevicceca/mobilne-aplikacije-sit

Za dodatna pitanja možete se obratiti asistentima:

- Svetlana Antešević (svetlanaantesevic@uns.ac.rs)
- Jelena Matković (matkovic.jelena@uns.ac.rs)