OSNOVE RAZVOJA WEB I MOBILNIH APLIKACIJA

LV 8: Google Firebase

Osijek, 2022.

1. PRIPREMA

U ovoj je vježbi prikazano je korištenje Firestore baze podataka u kombinaciji sa *RecyclerViewom*. U radu na vježbi ćemo napraviti prvi Firebase projekt i Firebase bazu podataka gdje ćemo spremiti osobe, a unutar same aplikacije ćemo omogućiti korisnicima da urede trenutne podatke o pojedinoj osobi ili ih obrišu.

1.2.3. Firebase

Firebase je platforma koja omogućava lako stvaranje i održavanje mobilnih i web aplikacija. Putem nje možete dobiti analitiku slučajnih padova vaše aplikacije (engl. *Crash*), mogućnost spremanja podataka u NoSQL bazu podataka, mogućnost lakše korisničke autentikacije i još mnoge druge alate koji olakšavaju izradu poslužiteljskog dijela mobilnih ili web aplikacija.

1.2.3.1. Cloud Firestore

Cloud Firestore je fleksibilna i skalabilna NoSQL baza podataka koja se nalazi na oblaku računala. Putem nje možete spremati ili sinkronizirati podatke za potrebe klijentske i poslužiteljske strane razvoja aplikacija. Omogućava sinkronizaciju u stvarnom vremenu što znači da korisnik ima pristup svojim podacima bez obzira ima li pristup internetu ili ne. Kako je Cloud Firestore NoSQL baza podataka to znači da se podaci spremaju kao dokumenti sa ključ/vrijednost načinom.

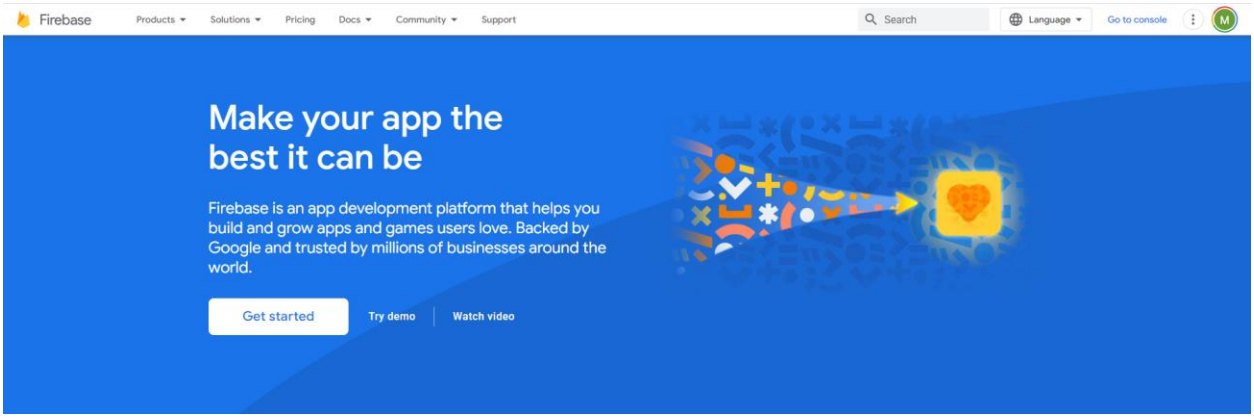
Više informacija o Cloud Firestoru i ostalim Firebase proizvodima moguće je pronaći na https://firebase.google.com/docs/firestore

2. RAD NA VJEŽBI

2.1. Stvaranje Firebase projekta

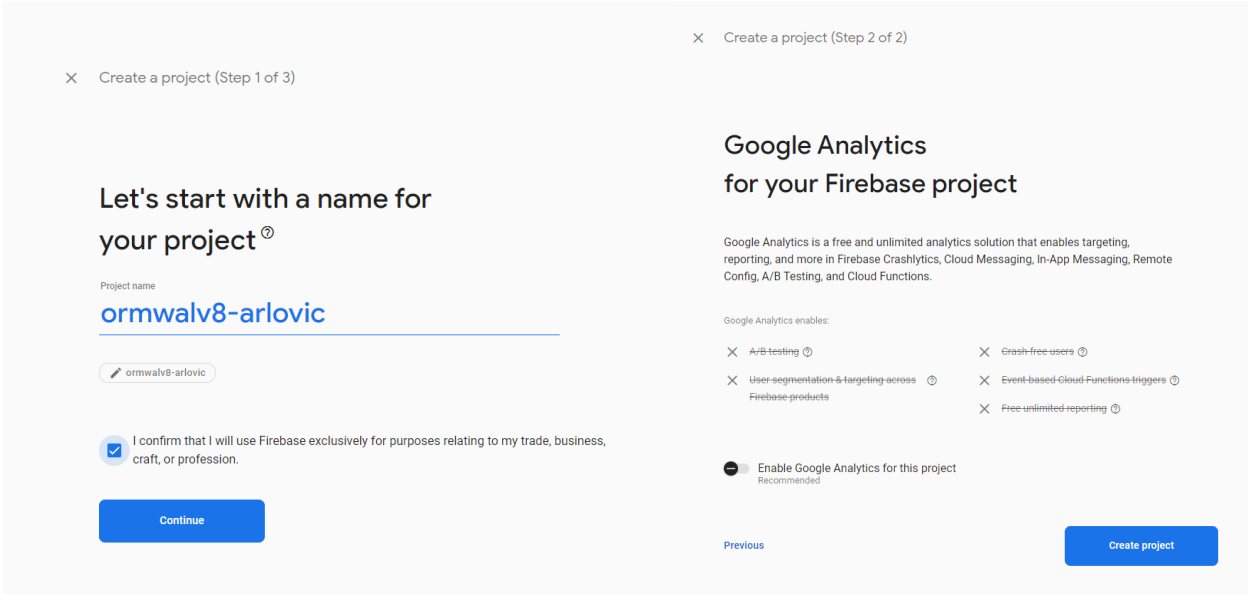
U ovome primjeru bi trebali realizirati jednu aktivnost s *RecyclerViewom* koji će prikazivati podatke koje ste spremili unutar *„persons“* kolekcije na Firestore bazi podataka, a koja sadrži: URL slike, ime i opis osobe. Za početak je potrebno stvoriti projekt na Firebase platformi (https://console.firebase.google.com) za to je potrebno kliknuti na „Go to console“ gumb koji se

nalazi na početnoj stranici Firebase web stranice. Početna stranica Firebase platforme prikazana je na slici 2.1.



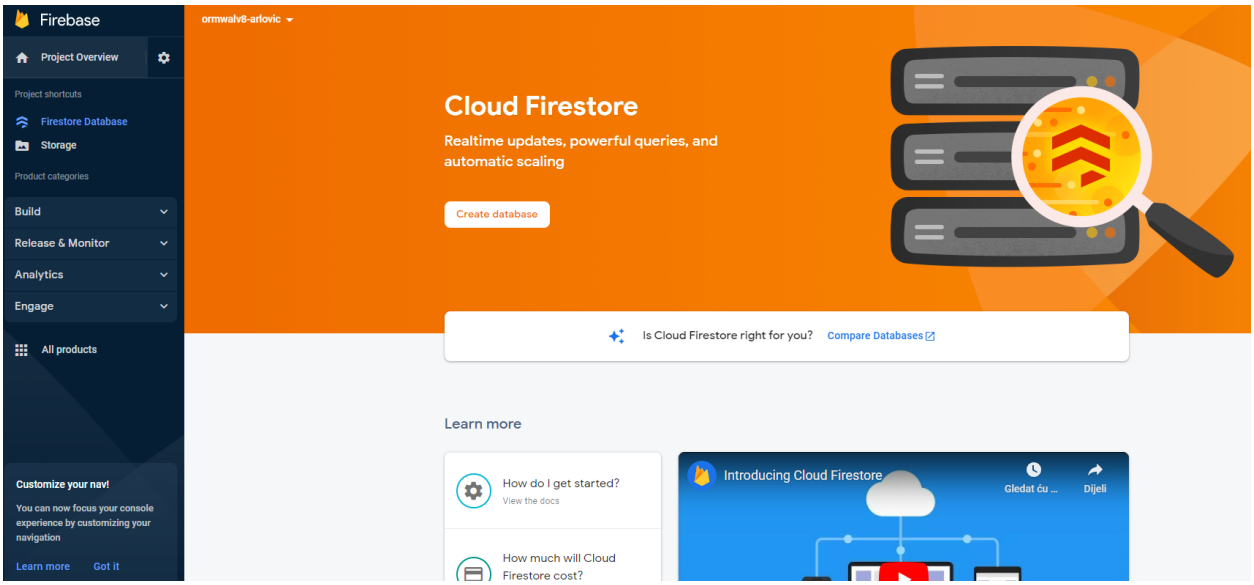
Slika 2.1. Prikaz početne stranice na Firebasea.

Nakon klika na gumb prikazati će vam se nekoliko dijaloga u kojima unosite informacije vezane za vaš projekt. Na početku trebate unijeti naziv projekta, a u našem slučaju to će biti *ormwalv8-**prezime***. Na drugom zaslonu potrebno je ugasiti suglasnost da ćete koristiti Google Analytics jer taj dio Firebasea nećemo koristiti. Na slikama 2.2 prikazano je stvaranje projekta na Firebase platformi.



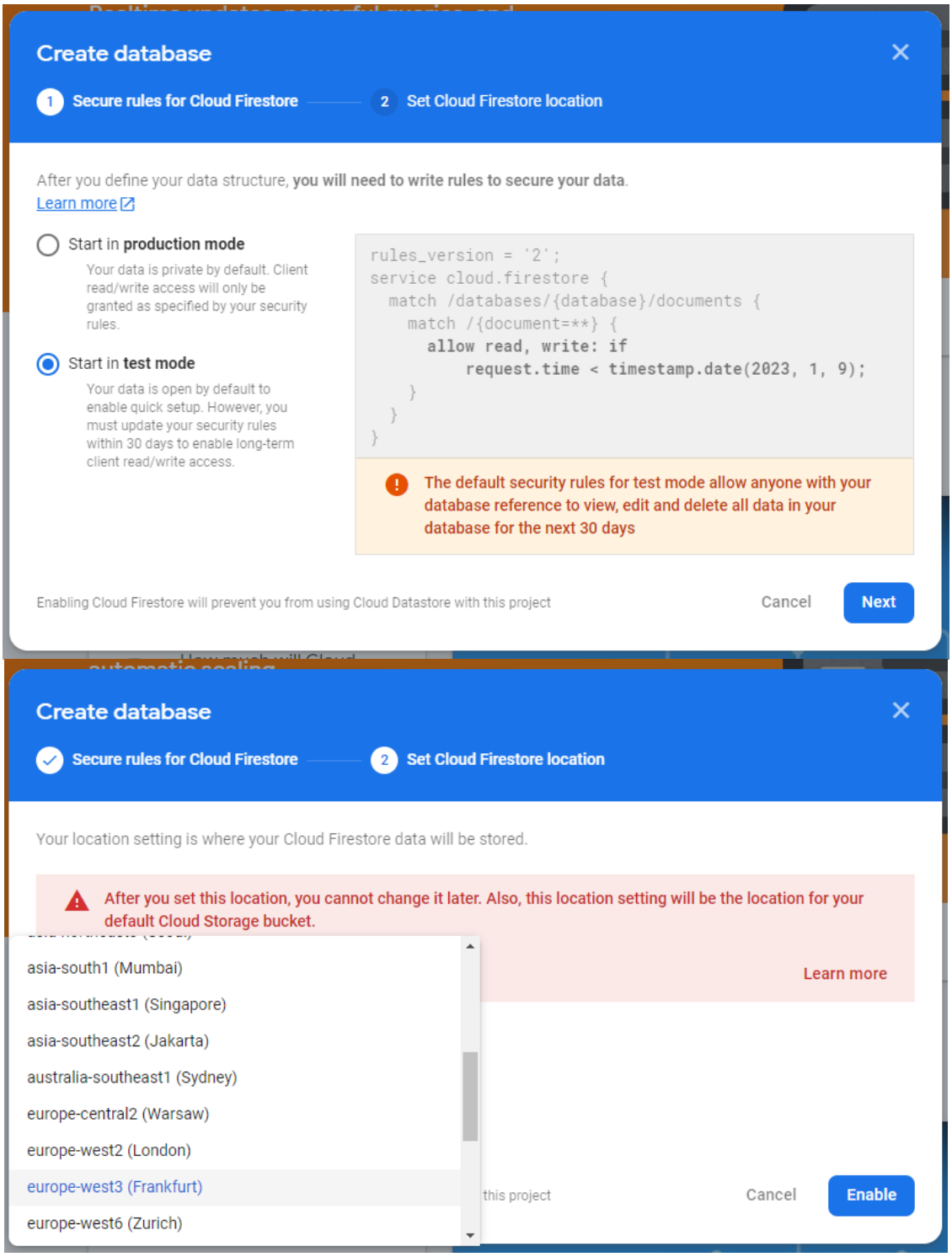
Slika 2.2. Prikaz kreiranja projekta na Firebase platformi.

S lijeve strane unutar menija nalazi se *Build* gumb koji će otvoriti padajući izbornik unutar kojeg će se nalaziti *Firestore Database*. Na slici 2.3 prikazana je stranica kada se klikne Firestore Database gumb.



Slika 2.3. Prikaz Firestore Database nadzorna ploča.

Klikom na *„Create database“* gumb otvoriti će vam se dijalog za kreiranje Firestore baze podataka. U prvom dijelu kreiranja potrebno je odabrati mod u kojem će se pokretati vaša baza podataka. Za potrebe laboratorijskih vježbi mi ćemo koristiti test mod. Taj mod omogućava pristup vašoj bazi podataka svim korisnicima koji imaju pristupne podatke, ali tada će vaša baza podataka biti dostupna samo 30 dana nakon kojih morate promijeniti sigurnosne postavke. Na slici 2.4 prikazano je kreiranje Firestore baze podataka. Na drugom zaslonu potrebno je odabrati lokaciju poslužitelja na kojem se nalazi Firestore baza podataka. Potrebno je odabrati **Frankfurt** kao lokaciju poslužitelja, a on se nalazi u *Regional* kategoriji.



Slika 2.4. Prikaz kreiranja Firestore baze podataka.

2.2. Model i RecyclerView

Nakon završetka postavljanja Firestore baze podataka, potrebno je prema znanjima stečenim u prošloj laboratorijskoj vježbi, instalirati Glide biblioteku. Nakon instaliranja Glide biblioteke potrebno je napraviti model za osobe koje ćemo spremati u bazu podataka, ali i prikazivati na *RecyclerViewu*. Na slici 2.5. prikazan je programski kod za model našeg podatka.

```
data class Person(
    val imageUrl: String,
    val name: String,
    val description: String
)
```

Slika 2.5. Prikaz programskog koda za model podataka.

Nakon definiranog modela podataka potrebno je definirati izgled izgled svake stavke u RecyclerViewu. Izgled je sličan izgledu iz prošle laboratorijske vježbe, ali umjesto *TextViewa* ćemo staviti *EditTextove* i dodati ćemo još dva gumba s kojima ćemo brisati podatke u bazi podataka ili ih urediti. Programski kod za izgled stavki u RecyclerViewu prikazan je na slici 2.6. Vaš izgled treba biti što sličniji izgledu na slici 2.7.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <ImageView
        android:id="@+id/personImage"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_marginStart="16dp"
        android:layout_marginTop="16dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/ic_launcher_background" />

    <EditText
        android:id="@+id/personName"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="4dp"
        android:layout_marginEnd="16dp"
        android:text="TextView"
        app:layout_constraintEnd_toStartOf="@+id/deleteButton"
        app:layout_constraintStart_toEndOf="@+id/personImage"
        app:layout_constraintTop_toTopOf="@+id/personImage" />
```

```xml
    <EditText
        android:id="@+id/personDescription"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        android:text="TextView"
        app:layout_constraintEnd_toStartOf="@+id/deleteButton"
        app:layout_constraintStart_toEndOf="@+id/personImage"
        app:layout_constraintTop_toBottomOf="@+id/personName" />

    <ImageButton
        android:id="@+id/deleteButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:backgroundTint="@color/purple_700"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/editButton"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/ic_baseline_delete_24" />

    <ImageButton
        android:id="@+id/editButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="16dp"
        android:backgroundTint="@color/purple_700"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/ic_baseline_edit_24" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Slika 2.6. Prikaz programskog koda za izgled stavki u RecyclerViewu.

Slika 2.7. Prikaz izgleda stavki u RecyclerViewu.

Prema znanjima stečenim u prošloj laboratorijskoj vježbi potrebno je napisati programski kod za RecyclerView adapter. On će biti uveliko sličan kao i *PersonRecyclerView* adapter iz prošle laboratorijske vježbe, ali ćemo u funkciji *bind* dodati i *setOnClickListener* za nove gumbove koje smo dodali. Programski kod za novi *PersonRecyclerView* adapter prikazan je na slici 2.8.

```
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ImageButton
import android.widget.ImageView
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.bumptech.glide.Glide

enum class ItemClickType {
    EDIT,
    REMOVE,
}

class PersonRecyclerAdapter(
    val items: ArrayList<Person>,
    val listener: ContentListener
): RecyclerView.Adapter<RecyclerView.ViewHolder>()  {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
RecyclerView.ViewHolder {
        return PersonViewHolder(
            LayoutInflater.from(parent.context).inflate(R.layout.recycler_item,
parent, false)
        )
    }

    override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
        when (holder) {
            is PersonViewHolder -> {
                holder.bind(position, listener, items[position])
            }
        }
    }

    override fun getItemCount(): Int {
        return items.size
    }

    fun removeItem(index: Int) {
        items.removeAt(index)
        notifyItemRemoved(index)
            notifyItemRangeChanged(index, items.size)
    }
```

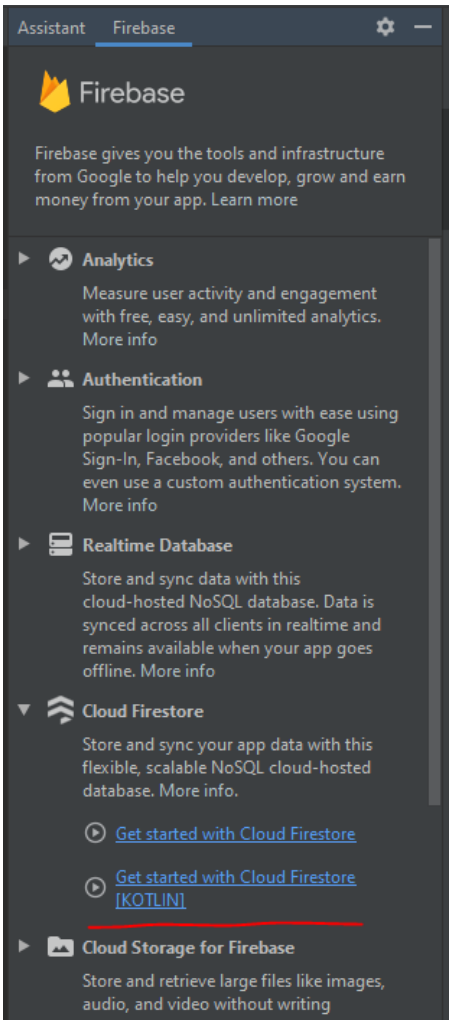
```
class PersonViewHolder(val view: View): RecyclerView.ViewHolder(view) {
        private val personImage =
view.findViewById<ImageView>(R.id.personImage)
        private val personName =
view.findViewById<TextView>(R.id.personName)
        private val personDescription =
view.findViewById<TextView>(R.id.personDescription)
        private val editBtn =
view.findViewById<ImageButton>(R.id.editButton)
        private val deleteBtn =
view.findViewById<ImageButton>(R.id.deleteButton)

        fun bind(
            index: Int,
            listener: ContentListener,
            person: Person
        ) {

Glide.with(view.context).load(person.imageUrl).into(personImage)
            personName.text = person.name
            personDescription.text = person.description

            editBtn.setOnClickListener {
                person.name = personName.text.toString()
                person.description = personDescription.text.toString()
                listener.onItemButtonClick(index, person,
ItemClickType.EDIT)
            }

            deleteBtn.setOnClickListener {
                listener.onItemButtonClick(index, person,
ItemClickType.REMOVE)
            }
        }
    }

    interface ContentListener {
        fun onItemButtonClick(index: Int, person: Person, clickType:
ItemClickType)
    }
}
```

Slika 2.8. Prikaz programskog koda za Adapter.

Zbog gumbova unutar *RecyclerView* stavki moramo implementirati prilagođen događaj koji smo definirali unutar *PersonRecyclerAdapter* klase. Kada korisnik klikne jedan od gumbova aplikacija će poslati događaj svim klasama koji osluškuju taj događaj. U *ItemClickType* enumu smo definirali dva događaja *EDIT* i *REMOVE*. I unutar *setOnClickListenera* koji se nalaze unutar *bind* metode okidamo *onItemButtonClick* događaj. U sljedećem poglavlju ćemo postaviti bazu podataka i odrediti što će *MainActivity* raditi kada se pošalje događaj iz *PersonRecyclerAdaptera*.

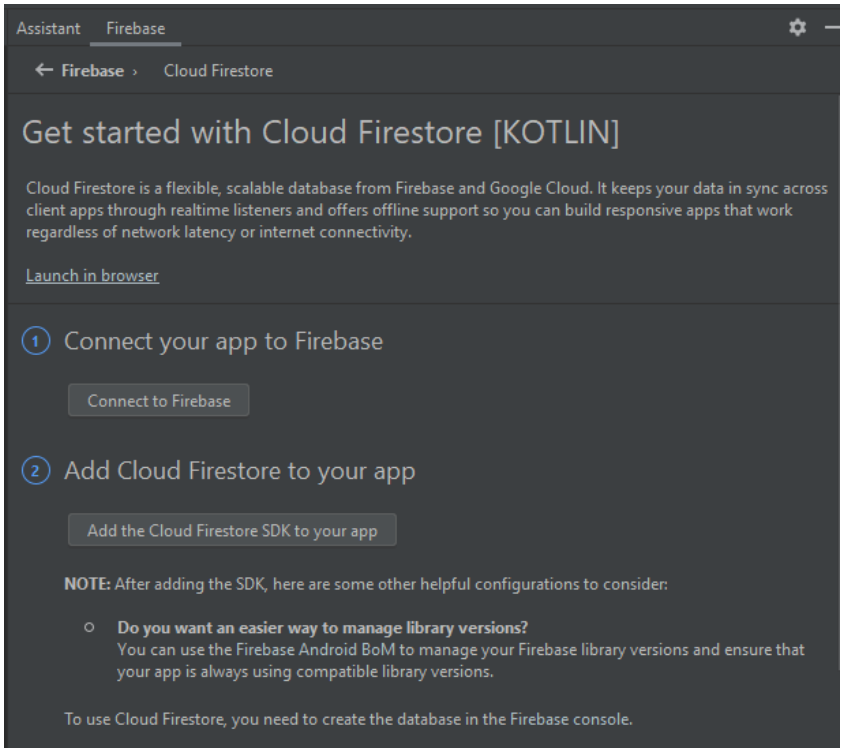
2.3. Dohvaćanje i spremanje podataka na Firebase Firesore baze podataka

Nakon završetka pisanja programskog koda za model podataka i programskog koda za RecyclerView adapter potrebno je povezati naš Android projekt s Firebase platformom. Unutar Android studija je potrebno otvoriti Firebase odjeljak tako da idete: Tools → Firebase. Sa strane će se pokazati Firebase odjeljak gdje možete odabrati Firebase proizvod koji želite implementirati u svoju Android aplikaciju. Za potrebe ovih vježba koristiti će se Cloud Firestore baza podataka. Na slici 2.9. prikazan je odjeljak za postavljanje Firebase proizvoda unutar Android aplikacije.



Slika 2.9. Prikaz Firebase odjeljka unutar Android studija.

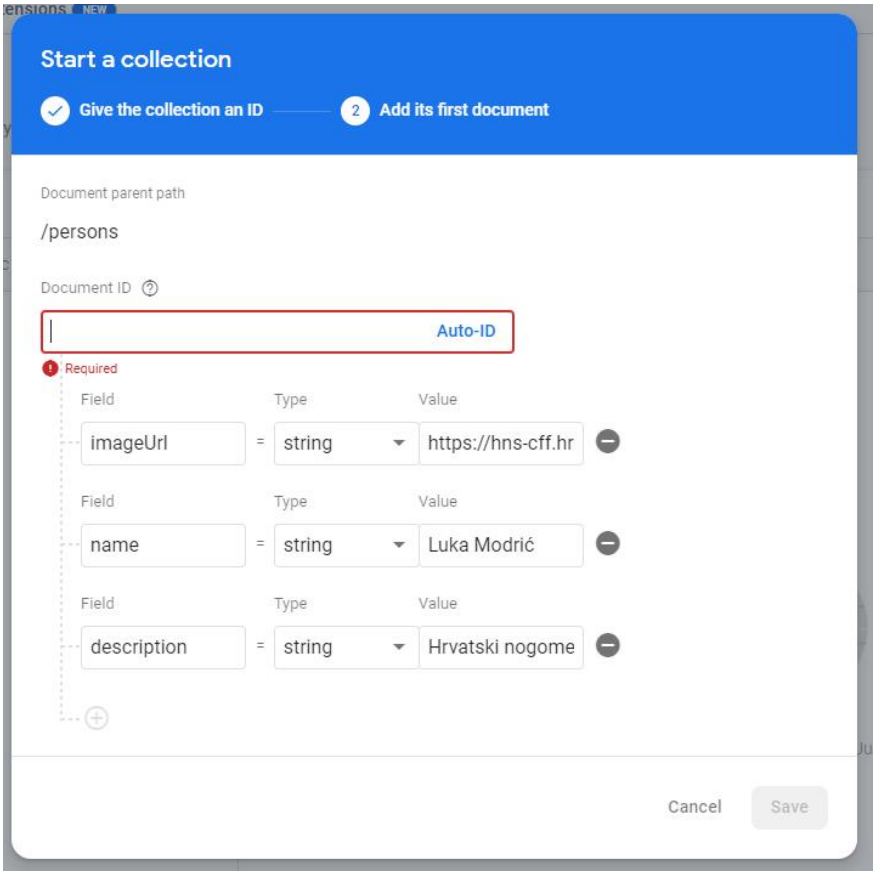
Kada kliknete na „*Getting started with Cloud Firestore [KOTLIN]*“ otvara se novi prozor s primjerima korištenja Cloud Firestorea i koracima za implementaciju Cloud Firestorea. Na slici 2.10. prikazan je novi prozor s koracima implementacije Cloud Firestorea unutar Android aplikacije.



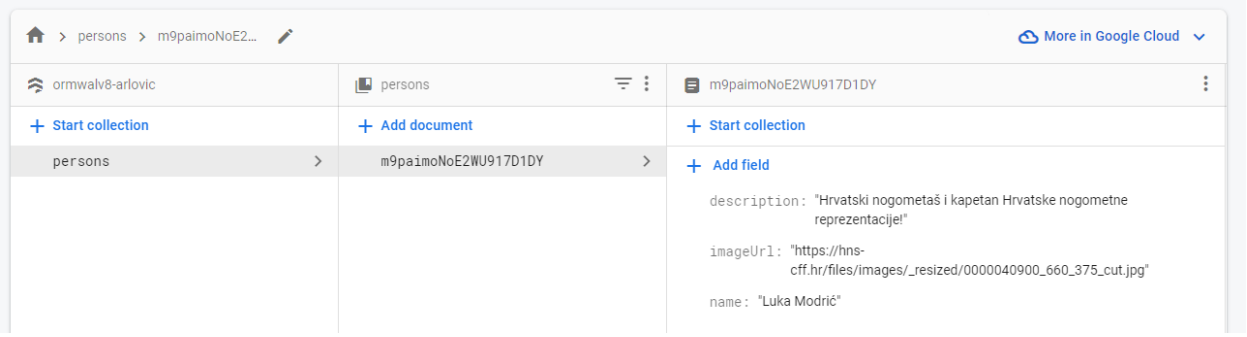
Slika 2.10. Prikaz koraka za implementaciju Cloud Firestorea unutar Android aplikacije.

Za početak je potrebno kliknuti gumb na kojem piše „*Connect to Firebase*“ kako bi spojili vaš Android projekt s Firebase projektom. Otvorit će se Firebase konzola s popisom vaših Firebase projekata i potrebno je odabrati projekt kojeg ste stvorili na početku primjera. Kada se završi sinkronizacija projekata potrebno je kliknuti na drugi gumb kako bi dodali Firebase SDK u vašu mobilnu aplikaciju. Kada kliknete „Add the Cloud Firestore SDK to your app“ odaberite „Accept Changes“ kako bi omogućili Android Studiju da automatski ažurira vaše Gradle datoteke. U ovome koraku će čarobnjak automatski postaviti sve potrebne biblioteke i pripremiti vaš projekt za korištenje Cloud Firestorea. Cloud Firestore je noSQL baza podataka i svoje podatke sprema u kolekcije i dokumente. Stoga je potrebno napraviti jednu kolekciju pod nazivom „*persons*“ i dodati nekoliko dokumenata s nasumičnim ID-om (možete kliknuti na gumb za automatsko generiranje nasumičnog ID-a). A svi dokumenti moraju imati podatke s ključevima: *name,*

imageUrl, description. Kako bi dodali novi dokument potrebno je kliknuti na „Add document“ gumb kako bi dodali nove podatke u vašu bazu podataka. Na slici 2.12. prikazano je dodavanje novih podataka u bazu podataka. Postavljeni podaci u Cloud Firestore bazi podataka su prikazani na slici 2.12.



Slika 2.11. Prikaz dodavanja novih podataka u bazu podataka.



Slika 2.12. Prikaz postavljenih testnih podataka na Cloud Firestoreu.

Nakon postavljene baze podataka potrebno je urediti *MainActivity* klasu kako bi se na početku aplikacije dohvatili svi podaci unutar baze podataka i prikazali se unutar RecyclerViewa. Osim

AppCompatActivity() MainActivity klasa nasljeđuje i osluškivača događaja *PersonRecyclerAdapter.ContentListener* koji smo napisali u *PersonRecyclerAdapter* klasi.

```
class MainActivity : AppCompatActivity(),
PersonRecyclerAdapter.ContentListener {
    private val db = Firebase.firestore
    private lateinit var recyclerAdapter: PersonRecyclerAdapter

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val recyclerView = findViewById<RecyclerView>(R.id.personList)
        db.collection("persons")
            .get()
            .addOnSuccessListener { result ->
                val personList = ArrayList<Person>()
                for (data in result.documents) {
                    val person = data.toObject(Person::class.java)
                    if (person != null) {
                        person.id = data.id
                        personList.add(person)
                    }
                }

                recyclerAdapter = PersonRecyclerAdapter(personList,
this@MainActivity)
                recyclerView.apply {
                    layoutManager = LinearLayoutManager(this@MainActivity)
                    adapter = recyclerAdapter
                }
            }
            .addOnFailureListener { exception ->
                Log.w("MainActivity", "Error getting documents.",
exception)
            }
    }

    override fun onItemButtonClick(index: Int, person: Person, clickType:
ItemClickType) {
        if (clickType == ItemClickType.EDIT) {
            db.collection("persons")
                .document(person.id)
                .set(person)
        }
        else if (clickType == ItemClickType.REMOVE) {
            recyclerAdapter.removeItem(index)
            db.collection("persons")
                .document(person.id)
                .delete()
        }
    }
}
```

Slika 2.13. Prikaz programskog koda za *MainActivity*.

Potrebno je definirati dvije globalne varijable koje će spremati instancu Firebase baze podataka i instancu adaptera za *RecyclerView*. Unutar *onCreate* metode je potrebno pronaći instancu RecyclerViewa i nakon toga dohvatiti sve podatke iz „persons“ kolekcije i unutar *addOnSuccessListenera* osluškivača događaja postaviti sve podatke u model i na kraju prikazati ih u *RecyclerViewu*. Zbog toga što se nasljeđuje *PersonRecyclerAdapter.ContentListener* potrebno je overrideati onItemButtonClick metodu koja će se pozvati svaki puta kada korisnik klikne gumbove u pojedinoj stavki u RecyclerViewu. Ako je *clickType == ItemClickType.EDIT* samo će se ažurirati podaci u bazi podataka, a ako je *clickType == ItemClickType.REMOVE* obrisati će se dokument pod odabranim ID-om, ali i pozvati će se *removeItem* metoda unutar adaptera koja će obrisati element unutar *items* liste i ažurirati *RecyclerView*. Na slici 2.13. prikazan je programski kod za *MainActivity* klasu. Naposlijetku potrebno je dodati RecyclerView element unutar *activity_main.xml* datoteke koja diktira izgled glavne aktivnosti.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/personList"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Slika 2.14. Prikaz programskog koda za *activity_main.xml*.

3. ZADACI ZA SAMOSTALAN RAD

1. Potrebno je napraviti Android aplikaciju koja će na temelju unosa iz *EditText* elemenata spremiti unesene podatke u model, prikazati ih na RecyclerViewu i dodati ih kao novi dokument u bazi podataka.

Slika 3.1. Prikaz izgleda *RecyclerViewa* za rješenje zadatka 1.