OSNOVE RAZVOJA WEB I MOBILNIH APLIKACIJA

LV 7: Recycler View i REST API

Osijek, 2022.

1. PRIPREMA

U ovoj je vježbi prikazano korištenje dohvata podataka sa REST API-ja pomoću Retrofita i njihov prikaz unutar *RecyclerViewa*.

1.1. RecyclerView

RecyclerView je UI komponenta koja je nastala kao proširenje za *ListView* komponentu, a koja prikazuje listu sadržaja. *RecyclerView* delegira posao rasporeda elemenata *LayoutManageru*, pa je tako moguće postići pomicanje sadržaja vertikalno i horizontalno. Elementi koji su izašli iz prikaza se ponovno recikliraju s novim sadržajem i prikazuju korisniku na zaslonu. Taj način prikaza elemenata značajno umanjuje zahtjeve prema računalnim resursima. *RecyclerView* zahtjeva korištenje *ViewHolder* oblikovanog obrasca koji omogućuje manje poziva *findViewById* metodi nad *View* objektima što dodatno ubrzava rad. Kod rada s *RecyclerViewom* nužno je napisati vlastitu adapter klasu koja će omogućiti prilagodbu prikaza sadržaja iz objekata modela u *View* objekte. Za razliku od *ListViewa*, *RecyclerView* omogućuje bolju interakciju s elementima iz liste, animaciju pojedinih elemenata, prilagodbu razmaka između elemenata i slično. Njegovo postavljanje je složenije u odnosu na *ListView*.

ViewHolder je klasa koja predstavlja jedan element unutar *RecyclerViewa* i upravo se ona koristi prilikom recikliranja elemenata. *RecyclerView* će napraviti onoliko instanci *ViewHoldera* koliko je potrebno da bi se korisnički ekran popunio podacima. Za sve podatke koji nisu trenutno prikazani na ekranu će se reciklirati već instancirani *ViewHolder* s novim sadržajem. Prilikom stvaranja *ViewHolder* objekta potrebno je kroz konstruktor poslati *View* objekt odnosno već napisani izgled objekta u XML-u.

Adapter je klasa koja se brine o stvaranju ViewHoldera kao i držanju podataka koji će se prikazivati unutar RecyclerViewa. Postoje tri sistemske metode koje se moraju upotrijebiti kako bi adapter funkcionirao. *getItemCount* je metoda koja vraća ukupni broj elemenata koji se nalaze u RecyclerViewu. Metoda koja je zadužena za kreiranje i recikliranje ViewHoldera naziva se *onCreateViewHolder*, a treća metoda naziva se *onBindViewHolder* i služi za povezivanje *ViewHoldera* i podataka koji se nalaze u adapteru.

1.2. REST API

REST arhitektura pomoću HTTP protokola dijeli podatke kroz mrežu, a oni mogu biti medijski ili tekstualni podaci. Za razmjenu podataka koriste transportne jezike poput JSON i XML.

1.2.1. JSON

JSON (engl. *JavaScript Object Notation*) je transportni jezik koji na efikasan način prenosi podatke. Podaci koje JSON prenosi su lako čitljivi ljudima, a istovremeno jednostavan za generiranje, parsiranje i obradu na računalima. Primjer izgleda JSON podataka se nalazi na slici 1.1.

Slika 1.1. Prikaz JSON podatka.

JSON podatak je niz objekata ili polja koji sadržavaju parove ključ/vrijednost.

1.2.2. Retrofit

Retrofit je REST (engl. *Representational state transfer*) klijent građen nad OkHttp klijentom dizajniran za Android i Javu. Koristi anotacije kako bi opisao različite HTTP zahtjeve, pruža mogućnost kao što su dinamički parametri unutar URL-a, parametre upita, prilagođena zaglavlja, skidanje i učitavanje datoteka, pisanje lažnih odgovora (engl. *Mocking responses*). Omogućava sinkrone i asinkrone zahtjeve, a samostalno brine o sinkronizaciji, upravljaju nitima, keširanju, učitavanju, itd. Za upravljanje HTTP zahtjevima koristi OkHttp biblioteku koja je obavezna uz

Retrofit, omogućuje jedinstveno definiranje krajnjih točaka komunikacije putem HTTP protokola (engl. *endpoint*). Koristeći Retrofit u kombinaciji sa Gson pretvaračem za JSON ili XML pretvaračem za XML prilično je lako podatke koji se nalaze na nekom Web servisu u JSON ili XML formatu dohvatiti i pretvoriti u POJO (engl. *Plain Old Java Object*). Retrofit automatski serijalizira JSON podatke koristeći Gson pretvarač i POJO koji mora biti unaprijed definiran po JSON strukturi, takav POJO objekt može sadržavati sve atribute koje ima JSON ili samo one koji se žele dohvatiti. Za izvršavanje Retrofit zahtjeva potrebna je model klasa koja se koristi za serijalizaciju odgovora, sučelje u kojem je definiran tip HTTP zahtjeva te Retrofit instancu koja se koristi za izvršavanje HTTP zahtjeva definiranog u sučelju. Svako sučelje predstavlja kolekciju različitih API poziva, svaka metoda unutar sučelja koja se želi koristi kao HTTP zahtjev mora sadržavati anotaciju zahtjeva koji obavlja.

2. RAD NA VJEŽBI

2.1. Stvaranje prvog RecyclerViewa

U ovome primjeru bi trebali realizirati jednu aktivnost s *RecyclerViewom* koji će prikazivati podatke koje ste vi spremili unutar *Persons* liste, a koja sadrži: URL slike, ime i prezime, datum rođenja i adresu. Za početak je potrebno stvoriti model podataka odnosno data klasu unutar koje će se spremiti osobe. Izgled modela klase prikazan je na slici 2.1.

```
data class Persons(
    var photoUrl: String,
    var name: String,
    var birthday: String,
    var address: String
)
```

Slika 2.1. Prikaz modela podataka u vidu data klase.

Sljedeći je korak riješiti izgled sadržaja koji će se nalaziti unutar RecyclerViewa, a kojeg će prikazivati Adapter klasa. Unutar *layouts* foldera stvorite novi *layout resource* (*New → Layout Resource File*). Unutar izgleda potrebno je opisati 1x *ImageView* i 3x *TextViewa*. Dodijelite im prigodne ID-eve. Programski kod koji opisuje izgled sadržaja prikazan je na slici 2.2.

Nakon postavke izgleda sadržaja potrebno je napisati programski kod za *RecyclerView* Adapter. Kao što je već spomenuto u pripremi vježbe adapter se brine da se sadržaji *RecyclerViewa* recikliraju ili pravilno prikažu. Adapter klasa zahtjeva preopteretiti *onCreateViewHolder*, *onBindViewHolder* i *getItemCount* metode. Unutar *onCreateViewHolder* metode se napuhuje (engl. *inflate*) izgled sadržaja pomoću *LayoutInflatera*. Dok se unutar *onBindViewHolder* metode se izvodi vezivanje pogleda (engl. *View binding*) između sadržaja unutar *RecyclerViewa* i *MainActivitya*.

U drugom dijelu Adapter klase je potrebno deklarirati i definirati ViewHolder klasu. *ViewHolder* klasa omogućava pozivanje skupocjene *findViewById* metode samo jednom po sadržaju *RecyclerViewa* i čuva te reference sve dok radi pogled u kojem je *RecyclerView* koji je povezan s *ViewHolder* klasom aktivan. Ukupan programski kod za Adapter klasu je prikazan na slici 2.3.

```
<?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/personPhoto"
        android:layout_width="128dp"
        android:layout_height="128dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:srcCompat="@tools:sample/avatars" />

    <TextView
        android:id="@+id/personName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="28dp"
        android:text="TextView"
        android:textAppearance="@style/TextAppearance.AppCompat.Body1"
        android:textSize="20sp"
        app:layout_constraintStart_toEndOf="@+id/personPhoto"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/personBirthday"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="8dp"
        android:text="TextView"
        android:textStyle="italic"
        app:layout_constraintStart_toEndOf="@+id/personPhoto"
        app:layout_constraintTop_toBottomOf="@+id/personName" />

    <TextView
        android:id="@+id/personAddress"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:text="TextView"
        android:textStyle="italic"
        app:layout_constraintStart_toEndOf="@+id/personPhoto"
        app:layout_constraintTop_toBottomOf="@+id/personBirthday" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Slika 2.2. Prikaz programskog koda koji opisuje izgled sadržaja RecyclerViewa.

```
class PersonRecyclerAdapter:
RecyclerView.Adapter<RecyclerView.ViewHolder>() {

    private var items: List<Persons> = ArrayList()

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
RecyclerView.ViewHolder {
        return PersonViewHolder(

LayoutInflater.from(parent.context).inflate(R.layout.person_layout, parent,
false)
        )
    }

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

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

    fun postItemsList(data: ArrayList<Persons>) {
        items = data
    }

    class PersonViewHolder constructor(
        itemView: View
    ): RecyclerView.ViewHolder(itemView) {
        private val photoImage: ImageView =
itemView.findViewById(R.id.personPhoto)
        private val personName: TextView =
itemView.findViewById(R.id.personName)
        private val personBirthday: TextView =
itemView.findViewById(R.id.personBirthday)
        private val personAddress: TextView =
itemView.findViewById(R.id.personAddress)

        fun bind(persons: Persons) {
            Glide
                .with(itemView.context)
                .load(persons.photoUrl)
                .into(photoImage)
            personName.text = persons.name
            personBirthday.text = persons.birthday
            personAddress.text = persons.address
        }
    }
}
```

Slika 2.3. Prikaz programskog koda za Adapter.

Sljedeći korak je povezivanje *RecyclerViewa* i *Adapter* klase unutar *MainAcitvity.kt*. Programski kod za *MainActivity.kt* prikazan je na slici 2.4.

```
class MainActivity : AppCompatActivity() {
    private lateinit var personAdapter: PersonRecyclerAdapter

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

        personAdapter = PersonRecyclerAdapter()
        personAdapter.postItemsList(setupData())
        initView()

    }

    private fun initView() {
        findViewById<RecyclerView>(R.id.personList).apply {
            layoutManager = LinearLayoutManager(this@MainActivity)
            adapter = personAdapter
        }
    }

    private fun setupData(): ArrayList<Persons> {
        val list = ArrayList<Persons>()
        list.add(
            Persons(
                "http://placeimg.com/640/480/people",
                "Pero Perić",
                "1971-09-14",
                "10657 Reichel Lodge Apt. 963"
            )
        )

        list.add(
            Persons(
                "http://placeimg.com/640/480/people",
                "Pero Perić123",
                "1971-09-20",
                "10657 Reichel Lodge Apt. 950"
            )
        )
        return list
    }
}
```

Slika 2.4. Prikaz programskog koda za *MainActivity.kt*.

Unutar *onCreate* metode se postavlja vrijednost *personAdapter* varijable s instancom adapter klase. Nakon toga je potrebno poslati *postItemsList* metodu koja će postaviti listu objekata *Persons* klase. Lista objekata se formira unutar *setupData* metode. Unutar initView metode pronalazi se RecyclerView unutar aktivnosti i definiraju se: *layoutManager* kao

LinearLayoutManager (daje isto ponašanje kao *ListView* komponenta) i *adapter* koji će biti jednak instanci *personAdapteru*.

2.2. Dohvaćanje podataka s REST API-ja

Ovaj primjer je nadopuna prošlog primjera samo će se ovdje podaci učitavati direktno iz vanjskog REST API-ja. Za potrebe ovoga primjera koristiti će se Faker API (https://fakerapi.it/en) koji je besplatni REST API za generiranje testnih podataka.

Potrebne su male izmjene unutar programskog koda model klase, ali je potrebno dodati novi programski kod kako bi olakšali danje izmjene programskog koda. Prvo na redu je napraviti izmjene unutar modela tako da se putem GSON biblioteke može lakše pretvoriti JSON podatke u POJO. Izgled programskog koda za modificiranu verziju *Persons data* klase je prikazan na slici 2.5.

```
data class ResponseData(
    val status: String,
    val code: Number,
    val total: Number,
    val data: ArrayList<Persons>
)

data class Persons(
    val firstname: String,
    val lastname: String,
    val email: String,
    val phone: String,
    val birthday: String,
    val gender: String,
    val address: Address,
    val website: String,
    val image: String
)

data class Address(
    val street: String,
    val streetName: String,
    val buildingNumber: Number,
    val city: String,
    val zipcode: String,
    val country: String,
    val county_code: String,
    val latitude: Double,
    val longitude: Double
)
```

Slika 2.5. Prikaz programskog rješenja za generiranje pravilnog POJO-a na temelju rezultata s Faker API-ja.

Nova model klasa je napisana tako da opisuje podatke koji se dobiju sa Faker API-ja. U sljedećem koraku potrebno je napraviti sučelje (engl. Interface) koje će voditi računa o krajnjim točkama (engl. Endpoints) REST API-ja koji se koristi. @GET anotacija govori Retrofitu da će treba koristiti GET poziv kako bi dobio podatke sa API-ja kao odgovor. Unutar zagrada od @GET antotacije se unosi link do podataka. Pozivom na *getPersons* metode *Retrofit* će poslati GET HTTP zahtjev serveru, a prihvaćene podatke vratiti će kao objekt *ResponseData* klase. Prikaz programskog koda za sučelje prikazano je na slici 2.6.

```
interface FakerEndpoints {
    @GET("/api/v1/persons?_quantity=10")
    fun getPersons(): Call<ResponseData>
}
```

Slika 2.6. Prikaz programskog koda za FakerEndpoints sučelje.

Sljedeći korak je kreiranje Kotlin objekta pod nazivom ServiceBuilder. Cilj tog objekta je pripremiti Retrofit za uspostavljanje poziva prema serveru. Prvo je potrebno stvoriti OkHttp klijent kojeg će Retrofit koristiti za pozive. Nakon toga je potrebno deklarirati i definirati retrofit builder objekt koji će sadržavati URL, pretvarač iz JSONa u POJO (GSON) i prethodno definiran OkHttpClient. buildService metoda se koristi kako bi povezali Retrofit builder objekt s prethodno napisanim sučeljem. Ta metoda vraća Retrofit objekt kojeg se može koristiti za druge pozive. Programski kod za ServiceBuilder je prikazan na slici 2.7.

```
object ServiceBuilder {
    private val client = OkHttpClient.Builder().build()

    private val retrofit = Retrofit.Builder()
        .baseUrl("https://fakerapi.it/")
        .addConverterFactory(GsonConverterFactory.create())
        .client(client)
        .build()

    fun<T> buildService(service: Class<T>): T{
        return retrofit.create(service)
    }
}
```

Slika 2.7. Prikaz programskog koda za ServiceBuilder objekt.

Za kraj potrebno je izmijeniti *MainActivity* klasu kako bi programski kod mogao automatski ažurirati *RecyclerView* na temelju podataka koje je Retrofit dohvatio s REST API-ja. Prvo je potrebno deklarirati i definirati *request* varijablu gdje se sprema stvoreni servis. Ista varijabla

koristi se za dohvaćanje Retrofit objekta koji se sprema u *call* varijablu. Pozivom *enque* metode se odašilje zahtjev ka REST API-ju, a unutar nje se mogu definirati odgovori na različita stanja zahtjeva u vidu metoda s povratnim pozivom. *onResponse* metoda se javlja kada se dobije odziv od strane API-ja i unutar nje se svi dohvaćeni podaci prosljeđuju adapter klasi. Programski kod za *MainActivity.kt* prikazan je na slici 2.8.

```
class MainActivity : AppCompatActivity() {

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

        val request =
ServiceBuilder.buildService(FakerEndpoints::class.java)
        val call = request.getPersons()

        call.enqueue(object : Callback<ResponseData> {
            override fun onResponse(call: Call<ResponseData>, response:
Response<ResponseData>) {
                if(response.isSuccessful) {
                    findViewById<RecyclerView>(R.id.personList).apply {
                        layoutManager =
LinearLayoutManager(this@MainActivity)
                        adapter =
PersonRecyclerAdapter(response.body()!!.data)
                    }
                }
            }
            override fun onFailure(call: Call<ResponseData>, t: Throwable)
{
                Log.d("FAIL", t.message.toString())
            }
        })
    }
}
```

Slika 2.8. Prikaz programskog koda preuređenog MainActivity.kt.

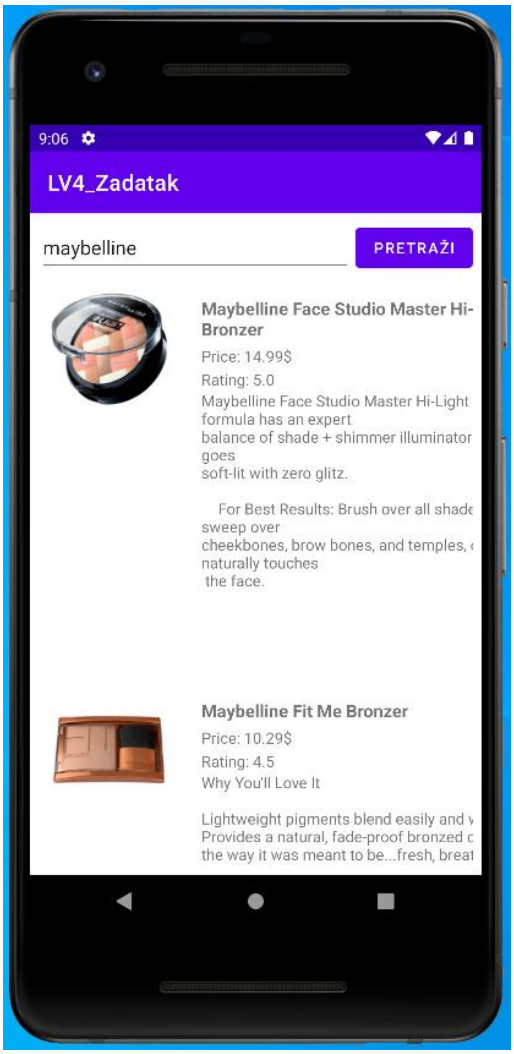
Kako su odrađene izmjene modela podataka potrebno je izvesti i izmjene *PersonRecyclerAdapter* klase. Programski kod novog *RecyclerAdaptera* je prikazan na slici 2.9.

```
class PersonRecyclerAdapter(private val items: List<Persons>):
RecyclerView.Adapter<RecyclerView.ViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
RecyclerView.ViewHolder {
        return PersonViewHolder(

LayoutInflater.from(parent.context).inflate(R.layout.person_layout, parent,
false)
        )
    }

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

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

    class PersonViewHolder constructor(
        itemView: View
    ): RecyclerView.ViewHolder(itemView) {
        private val photoImage: ImageView =
itemView.findViewById(R.id.personPhoto)
        private val personName: TextView =
itemView.findViewById(R.id.personName)
        private val personBirthday: TextView =
itemView.findViewById(R.id.personBirthday)
        private val personAddress: TextView =
itemView.findViewById(R.id.personAddress)

        fun bind(persons: Persons) {
            Glide
                .with(itemView.context)
                .load(persons.image)
                .into(photoImage)

            personName.text = "${persons.firstname} ${persons.lastname}"
            personBirthday.text = persons.birthday
            personAddress.text = persons.address.street
        }
    }
}
```

Slika 2.9. Prikaz programskog koda za adapter klasu.

3. ZADACI ZA SAMOSTALAN RAD

1. Potrebno je napraviti Android aplikaciju koja će na temelju unosa iz *EditText* elementa dohvaćati informacije o šminki sa https://makeup-api.herokuapp.com/. Primjer za dohvaćanje svih proizvoda brenda Maybelline je: https://makeup-api.herokuapp.com/api/v1/products.json?brand=maybelline. Rezultate pretraživanja je potrebno prikazati unutar RecyclerViewa koji ima prilagođen izgled (*layout*). Prilagođen izgled se treba sastojati od: 1x *ImageViewa* i 4x *TextViewa.* Dok se aktivnost treba sastojati od: 1x *EditText*, 1x *Button* i 1x *RecyclerView*. Nakon što korisnik klikne gumb „Pretraži“ aplikacija šalje zahtjev na poslužitelj putem Retrofita, a nakon toga pretvara JSON podatak u POJO.



Slika 3.1. Prikaz izgleda Fragmenata za rješenje zadatka 1.