Službeni šalabahter

ENTITY (@Entity)

```
@Entity(tableName = "table_name", foreignKeys = {
    @ForeignKey(entity = OtherEntity.class,
                parentColumns = "id",
                childColumns = "other_id",
                onDelete = ForeignKey.CASCADE)
})
public class MyEntity {
    @PrimaryKey(autoGenerate = true)
    private int id;

    private int other_id; // foreign key

    @NonNull
    @ColumnInfo(name = "name")
    private String name;

    @ColumnInfo(name = "value")
    private double value;

    // Konstruktor
    public MyEntity(@NonNull String name, double value) {
        this.name = name;
        this.value = value;
    }

    // Getteri i Setteri
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    @NonNull
    public String getName() { return name; }
    public void setName(@NonNull String name) { this.name = name; }

    public double getValue() { return value; }
    public void setValue(double value) { this.value = value; }
}
```

DAO (@Dao)

```
@Dao
public interface MyDao {

    // Osnovne operacije bez SQL-a
    @Insert
    void insert(MyEntity entity);

    @Update
    void update(MyEntity entity);

    @Delete
    void delete(MyEntity entity);

    // Prilagođeni upiti s @Query
    // QUERY FUNKCIJE: AGGREGATE, WHERE, ORDER BY, LIMIT
    // SELECT FUNKCIJE: COUNT, SUM, AVG, MIN, MAX
    @Query("SELECT * FROM table_name")
    LiveData<List<MyEntity>> getAll();

    @Query("SELECT * FROM table_name WHERE name = :name ORDER BY value DESC
LIMIT 1")
    LiveData<MyEntity> getByName(String name);

    @Query("DELETE FROM table_name WHERE id = :id")
    void deleteById(int id);

    @Query("SELECT MIN(value) FROM table_name")
    LiveData<Double> etMinValue();
}
```

DATABASE & PROVIDER

AppDatabase:

```
@Database(entities = {Entity.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract MyDao myDao();
}
```

DatabaseProvider (Singleton):

```
private static AppDatabase instance;

public static AppDatabase getInstance(Context context) {
    if (instance == null) {
        instance = Room.databaseBuilder(context, AppDatabase.class,
"db_name.db")
                .allowMainThreadQueries()
                .build();
    }
    return instance;
}
```

SINGLETON POMOĆNA KLASA & KORIŠTENJE BAZE

```
public class MyHelperClass {
    private static MyHelperClass instance;
    private DAO dao;

    private MyHelperClass(Context context) {
        AppDatabase db = DatabaseProvider.getInstance(context);
        this.dao = db.myDao();
    }

    public static synchronized MyHelperClass getInstance(Context context) {
        if (instance == null) instance = new MyHelperClass(context);
        return instance;
    }

    public void addMyClass(Entity e) {
        new Thread(() -> {
            dao.insert(e);
        }).start();
    }
    public void deleteMyClass(Entity e) {
        new Thread(() -> {
            dao.delete(e);
        }).start();
    }
    public LiveData<List<Entity>> getAll() {
        return dao.getAll();
    }
}
```

VIEWMODEL

```
public class SharedViewModel extends ViewModel {
    private MyHelperClass myHelperClass;
    private LiveData<List<Entity>> myClasses;

    public SharedViewModel(@NonNull Application application) {
        super(application);
        myHelperClass = MyHelperClass.getInstance(application);
        myClasses = myHelperClass.getAll();
    }

    public MyHelperClass getMyHelperClass() {
        return myHelperClass;
    }
    public LiveData<List<Entity>> getMyClasses() {
        return myClasses;
    }
}
```

ADAPTER (RecyclerView)

```
public class MyAdapter extends RecyclerView.Adapter<MyAdapter.ViewHolder> {
    private List<Entity> items;
    private final OnActionListener listener;

    public interface OnActionListener {
        void onButtonPressed(Entity item);
    }
    public MyAdapter(OnActionListener listener) {
        this.items = new ArrayList<>();
        this.listener = listener;
    }
    public void setItems(List<Entity> items) {
        this.items = items;
        notifyDataSetChanged();
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.item_id, parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        Entity item = items.get(position);
        holder.bind(item, listener);
    }
```

```
    @Override
    public int getItemCount() { return items.size(); }

    public static class ViewHolder extends RecyclerView.ViewHolder {
        TextView textView;
        Button btn;

        public ViewHolder(View itemView) {
            super(itemView);
            textView = itemView.findViewById(R.id.textView);
            btn = itemView.findViewById(R.id.btn);
        }

        void bind(Entity item, OnActionListener listener) {
            textView.setText(item.getKlasa());
            btn.setOnClickListener(v -> listener.onButtonPressed(item));
        }
    }
}
```

FRAGMENTI & KORIŠTENJE VIEWMODELA/ADAPTERA

```
public class myFragment extends Fragment {
    private SharedViewModel viewModel;
    private MyAdapter adapter;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle state) {
        View view = inflater.inflate(R.layout.fragment_my, container, false);

        viewModel = new
ViewModelProvider(requireActivity()).get(SharedViewModel.class);

        // Korištenje adaptera
        RecyclerView recycler = view.findViewById(R.id.recycler);
        adapter = new MyAdapter((item) -> { /* onButtonPressed */ });
        recycler.setAdapter(adapter);

        // Prati LiveData
        viewModel.getMyClasses().observe(getViewLifecycleOwner(), items -> {
            adapter.setItems(items);
        });

        return view;
    }
}
```

KORIŠTENJE FRAGMENATA

```
getSupportFragmentManager().beginTransaction()
    .replace(R.id.fragment_container, new myFragment())
    .commit();
```

TOAST & ALERTDIALOG

```
Toast.makeText(requireContext(), "Poruka", Toast.LENGTH_SHORT).show();

AlertDialog.Builder builder = new AlertDialog.Builder(requireContext());
EditText input = new EditText(requireContext());

builder.setTitle("Naslov");
builder.setView(input);
builder.setPositiveButton("OK", (dialog, which) -> { /* akcija na OK */ });
builder.setNegativeButton("Cancel", null);
builder.show();
```

LISTENERI

Button onClickListener:

```
button.setOnClickListener(v -> {
    // akcija na klik
});
```

CheckBox onCheckedChangeListener:

```
checkBox.setOnCheckedChangeListener((buttonView, isChecked) -> {
    // akcija na promjenu stanja
});
```

TextWatcher:

```
editText.addTextChangedListener(new TextWatcher() {
    @Override public void beforeTextChanged(CharSequence s, int start, int
count, int after) {}
    @Override public void onTextChanged(CharSequence s, int start, int before,
int count) {
        // akcija na promjenu teksta
    }
    @Override public void afterTextChanged(Editable s) {}
});
```