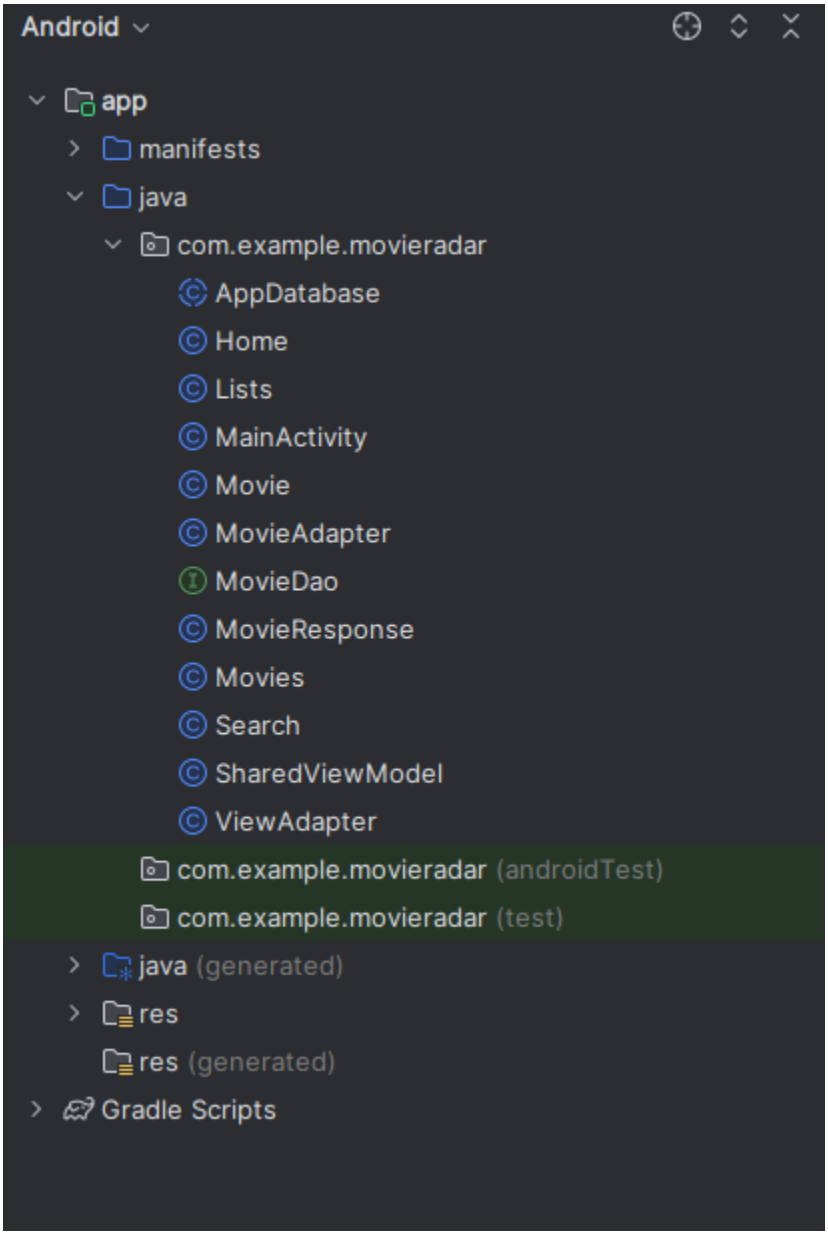
# Android bead

## Tartalom

## Osztályok:




Az oldal kialakítása Fragmantek betötése (tabok):(MainActivity,ViewAdapter)

Fragmanterk (Home, Search,Lists)

Adatbázis kialakításához használt osztályok:(MovieDao,AppDatabase,Movie)

Retrofit json kiolvasásához használt osztályok:(MovieResponse,Movies)

Fragmanterk közötti adat átvitel:(SharedViewModel)

## Az oldal kialakítása Fragmantek betötése (tabok):(MainActivity,ViewAdapter) :

```java
pager = findViewById(R.id.MoviePager);
ViewPager2 pager = findViewById(R.id.MoviePager);
TabLayout tabs = findViewById(R.id.TabMovies);

ViewAdapter adapter = new ViewAdapter( fragmentActivity: this);
//Ez a sor létrehoz egy új ViewAdapter objektumot, amely az oldalak tartalmát kezeli.
pager.setAdapter(adapter);
//Beállijuk a pager adapterét (ami kezeli)
tabs.addOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
    no usages
    @Override
    public void onTabSelected(TabLayout.Tab tab) {
        pager.setCurrentItem(tab.getPosition());
    }

    no usages
    @Override
    public void onTabUnselected(TabLayout.Tab tab) {
    }

    no usages
    @Override
    public void onTabReselected(TabLayout.Tab tab) {
    }
});
```

Amint a képen is látszik, létrehozzuk a **ViewAdapter** objektumot, majd ezt átadjuk a **ViewPager**-nek adapterként. Ennek eredményeképpen, amikor a **ViewPager** vált vagy kiválasztjuk a kívánt tabot, az adapter alapján betölti a megfelelő fragmentet.

## A Viewadapter:

```java
package com.example.movieradar;

import ...

2 usages
public class ViewAdapter  extends FragmentStateAdapter {

    1 usage
    public ViewAdapter(@NonNull FragmentActivity fragmentActivity) { super(fragmentActivity); }

    //A kapott positiontól függően betöltjük (létrehozzuk a fragmentet)
    1 usage
    @NonNull
    @Override
    public Fragment createFragment(int position) {
        switch (position) {
            case 0:
                return new Search() ;
            case 1:
                return new Home();
            case 2:
                return new Lists();
            default: return new Home();
        }
```

## Home:

**API Interfész**: A **MovieRequest** interfész tartalmaz egy @**GET** annotációval ellátott metódust, amely a **retrofit2** könyvtárat használja filmek lekérdezésére a The Movie Database (TMDB) API-jából.

```java
interface MovieRequest {
    @GET("3/discover/movie")
    Call<MovieResponse> getMovies(
            @Query("api_key") String apiKey,
            @Query("page") int page,
            @Query("with_genres") String gens,
            @Query("primary_release_year") String year
    );
}
```

**Felhasználói Felület Elemei**: A **Home** osztályban különböző felhasználói felület elemek vannak definiálva, mint például **TextView**-k és **ImageView** egy film címének, leírásának, megjelenési dátumának és poszterének megjelenítésére.

**Room Adatbázis**: A Room adatbázis inicializálása történik az **onCreateView** metódusban, amely lehetővé teszi a filmek tárolását helyileg.

```
db = Room.databaseBuilder(requireContext(), AppDatabase.class, "movies.db")
        .allowMainThreadQueries()
        .build();
movieDao = db.movieDao();
```

**ViewModel és Observerek**: A **ViewModel**-t használja a kiválasztott műfajok és év megfigyelésére. Amikor ezek az adatok megváltoznak, a **NextMovie** metódus hívódik meg, hogy új filmet jelenítsen meg.

```
viewModel.getSelectedGenres().observe(getViewLifecycleOwner(), new
Observer<List<Integer>>() {
    @Override
    public void onChanged(List<Integer> genres) {
        selectedGenres = genres;
        selectedGen = selectedGenres.toString().substring(1,
selectedGenres.toString().length() - 1);
        NextMovie();
    }
});

viewModel.getSelectedYear().observe(getViewLifecycleOwner(), new
Observer<String>() {
    @Override
    public void onChanged(String year) {
        selectedYear = year;
        NextMovie();
    }
});
```

**Következő Film Megjelenítése**: A **NextMovie** metódus API hívást indít, hogy lekérje a filmeket az adott oldalon, műfajban és évben. A válasz feldolgozása után a film adatai megjelennek a felhasználói felületen.

```
private void NextMovie() {
    Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("https://api.themoviedb.org/")
            .addConverterFactory(GsonConverterFactory.create())
            .build();

    MovieRequest movieApi = retrofit.create(MovieRequest.class);
    Call<MovieResponse> listCall =
movieApi.getMovies("da731e2507284961a1e0fe66b7824496", page_count, selectedGen,
selectedYear);

    listCall.enqueue(new Callback<MovieResponse>() {
        @Override
        public void onResponse(Call<MovieResponse> call,
Response<MovieResponse> response) {
            if (response.isSuccessful() && response.body() != null) {
                // Filmek feldolgozása és megjelenítése
            } else {
                Log.e("NetworkError", "Hiba a válaszban");
            }
        }

        @Override
        public void onFailure(Call<MovieResponse> call, Throwable t) {
            Log.e("NetworkError", "Hálózati hiba történt: " + t.getMessage());
            t.printStackTrace();
            Title.setText(t.getMessage());
        }
    });
}
```

**Film Mentése**: A **LikeMovie** metódus menti az aktuális filmet a helyi adatbázisba és betölti a következő filmet.

```
private void LikeMovie() {
    // API hívás és film mentése
}
```

**Adatbázis Ellenőrzés**: A **MovieId** és **saveCurrentMovie** metódusok ellenőrzik, hogy egy film már létezik-e az adatbázisban, és ha nem, akkor mentik azt.

```
private boolean MovieId(int movieId) {
    List<Movie> allMovies = movieDao.getAllMovies();
    // Film azonosításának ellenőrzése
}

private boolean saveCurrentMovie(int movieId, String posterPath) {
    // Film mentése az adatbázisba
}
```

## Search:

**Felhasználói Felület Elemei**: A **Search** osztályban különböző felhasználói felület elemek vannak definiálva, mint például **CheckBox**-ok a különböző műfajokhoz, egy **Spinner** az évek kiválasztásához, és egy **Button** a keresési feltételek beküldéséhez.

```
private CheckBox checkBoxAction, checkBoxAdventure, checkBoxAnimation,
checkBoxComedy, checkBoxCrime,
        checkBoxDrama, checkBoxFamily, checkBoxFantasy, checkBoxHorror,
checkBoxRomance;
private Spinner yearSpinner;
private Button submitButton;
```

**Műfajok ID-k**: Egy tömb tartalmazza a különböző műfajokhoz tartozó ID-kat.(TMBD api ezeket az id-ket használja)

```
private final int[] genreIds = {
        28, 12, 16, 35, 80,
        18, 10751, 14, 27, 10749
};
```

**ViewModel**: A **SharedViewModel** osztályt használják az adatok megosztására a különböző **Fragment**-ek között.

```
private SharedViewModel viewModel;
```

**Felhasználói Felület Elemeinek Inicializálása**: Az **onCreateView** metódusban inicializálják a felhasználói felület elemeit.

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

    checkBoxAction = view.findViewById(R.id.checkBoxAction);
    checkBoxAdventure = view.findViewById(R.id.checkBoxAdventure);
    checkBoxAnimation = view.findViewById(R.id.checkBoxAnimation);
    checkBoxComedy = view.findViewById(R.id.checkBoxComedy);
    checkBoxCrime = view.findViewById(R.id.checkBoxCrime);
    checkBoxDrama = view.findViewById(R.id.checkBoxDrama);
    checkBoxFamily = view.findViewById(R.id.checkBoxFamily);
    checkBoxFantasy = view.findViewById(R.id.checkBoxFantasy);
    checkBoxHorror = view.findViewById(R.id.checkBoxHorror);
    checkBoxRomance = view.findViewById(R.id.checkBoxRomance);

    yearSpinner = view.findViewById(R.id.yearSpinner);
    populateYearSpinner();

    submitButton = view.findViewById(R.id.submitButton);
    submitButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            handleSubmit();
        }
    });

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

    return view;
}
```

**Évek Spinner Kitöltése**: A **populateYearSpinner** metódus feltölti a **Spinner**-t a jelenlegi évtől kezdve egészen 1950-ig.

```
private void populateYearSpinner() {
    List<String> years = new ArrayList<>();
    int currentYear = Calendar.getInstance().get(Calendar.YEAR);
    years.add("");
    for (int i = currentYear; i >= 1950; i--) {
        years.add(String.valueOf(i));
    }
    ArrayAdapter<String> adapter = new ArrayAdapter<>(getContext(),
android.R.layout.simple_spinner_item, years);
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    yearSpinner.setAdapter(adapter);
}
```

**Keresési Feltételek Beküldése**: A **handleSubmit** metódus begyűjti a felhasználó által kiválasztott műfajokat és évet, majd ezeket átadja a **SharedViewModel**-nek.

```
private void handleSubmit() {
    List<Integer> selectedGenres = new ArrayList<>();

    if (checkBoxAction.isChecked()) selectedGenres.add(genreIds[0]);
    if (checkBoxAdventure.isChecked())
selectedGenres.add(genreIds[1]);
    if (checkBoxAnimation.isChecked())
selectedGenres.add(genreIds[2]);
    if (checkBoxComedy.isChecked()) selectedGenres.add(genreIds[3]);
    if (checkBoxCrime.isChecked()) selectedGenres.add(genreIds[4]);
    if (checkBoxDrama.isChecked()) selectedGenres.add(genreIds[5]);
    if (checkBoxFamily.isChecked()) selectedGenres.add(genreIds[6]);
    if (checkBoxFantasy.isChecked()) selectedGenres.add(genreIds[7]);
    if (checkBoxHorror.isChecked()) selectedGenres.add(genreIds[8]);
    if (checkBoxRomance.isChecked()) selectedGenres.add(genreIds[9]);

    int[] genreArray = new int[selectedGenres.size()];
    for (int i = 0; i < selectedGenres.size(); i++) {
        genreArray[i] = selectedGenres.get(i);
    }

    String selectedYear = yearSpinner.getSelectedItem().toString();

    viewModel.setSelectedGenres(selectedGenres);
    viewModel.setSelectedYear(selectedYear);

    Toast.makeText(getContext(), "Go to Home page" + selectedYear,
Toast.LENGTH_LONG).show();
}
```

# Lists

## Változók definiálása

```
private RecyclerView recyclerView;
private MovieAdapter adapter;
private AppDatabase db;
private MovieDao movieDao;
private List<Movie> movieList;
private boolean isZoomed = false;
private boolean Del_Zoom = false;
private ImageView fullscreenImageView;
private Button button;
```

recyclerView: A filmek listájának megjelenítésére szolgáló RecyclerView.

adapter: A RecyclerView adapter, amely kezeli a filmek adatainak megjelenítését.

db: A Room adatbázis referencia.

movieDao: A DAO (Data Access Object) az adatbázis műveletekhez.

movieList: A megjelenítendő filmek listája.

isZoomed: Az állapot jelzésére, hogy a kép kinagyított-e.

Del_Zoom: Az állapot jelzésére, hogy a gomb törlés vagy nagyítás funkciót lát el.

fullscreenImageView: A kinagyított kép megjelenítésére szolgáló ImageView.

button: A funkció váltására szolgáló gomb.

## onCreateView metódus

```
@Nullable
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
    // Inflate the layout for this fragment
    View view = inflater.inflate(R.layout.fragment_l_ists, container, false);

    recyclerView = view.findViewById(R.id.Mlist);
    fullscreenImageView = view.findViewById(R.id.fullscreenImageView);
    button = view.findViewById(R.id.Zoom_Delete);

    db = Room.databaseBuilder(requireContext(), AppDatabase.class, "movies.db")
            .allowMainThreadQueries()
            .build();
    movieDao = db.movieDao();
    movieList = movieDao.getAllMovies();
```

```
    adapter = new MovieAdapter(requireContext(), movieList, this);
    recyclerView.setLayoutManager(new GridLayoutManager(requireContext(), 2));
    recyclerView.setAdapter(adapter);


    fullscreenImageView.setOnClickListener(v -> {
        fullscreenImageView.setVisibility(View.GONE);
        recyclerView.setVisibility(View.VISIBLE);
        button.setVisibility(View.VISIBLE);
        isZoomed = false;
    });

    button.setOnClickListener(v -> {
        Del_Zoom = !Del_Zoom;
        if (Del_Zoom) {
            button.setText("Delete");
        } else {
            button.setText("Zoom");
        }
    });

    return view;
}
```

- A felület inicializálása, RecyclerView és ImageView beállítása.
- Adatbázis kapcsolat létrehozása és adatok lekérdezése.
- Adapter inicializálása és RecyclerView-hez csatolása.
- Kattintás események kezelése a kinagyításhoz és törlés funkcióhoz.

**refreshData és onItemClick metódusok**

```
private void refreshData() {
    movieList = movieDao.getAllMovies();
    adapter = new MovieAdapter(requireContext(), movieList, this);
    recyclerView.setLayoutManager(new GridLayoutManager(requireContext(), 2));
    recyclerView.setAdapter(adapter);
}

@Override
public void onItemClick(int position) {
    Movie movie = movieList.get(position);

    if (Del_Zoom) {
        movieDao.deleteMovieById(movie.id);
        Toast.makeText(requireContext(), "Movie deleted",
Toast.LENGTH_SHORT).show();
        refreshData();
    } else {
        if (isZoomed) {
            fullscreenImageView.setVisibility(View.GONE);
            recyclerView.setVisibility(View.VISIBLE);
```

```
            button.setVisibility(View.VISIBLE);
            isZoomed = false;
        } else {
            recyclerView.setVisibility(View.GONE);
            fullscreenImageView.setVisibility(View.VISIBLE);
            button.setVisibility(View.GONE);
            String baseUrl = "https://image.tmdb.org/t/p/original";
            Glide.with(requireContext())
                    .load(baseUrl + movie.posterPath)
                    .into(fullscreenImageView);
            isZoomed = true;
        }
    }
}
```

- **refreshData**: A filmek listájának frissítése az adatbázisból.
- **onItemClick**: Kattintás esemény kezelése - film törlése vagy kinagyítása a kiválasztott film adatainak megjelenítése alapján.

## MovieAdapter osztály

```
public class MovieAdapter extends
RecyclerView.Adapter<MovieAdapter.MovieViewHolder> {

    private Context context;
    private List<Movie> movies;
    private OnItemClickListener listener;

    public MovieAdapter(Context context, List<Movie> movies,
OnItemClickListener listener) {
        this.context = context;
        this.movies = movies;
        this.listener = listener;
    }
```

- **context**: A context referencia.
- **movies**: A filmek listája.
- **listener**: Kattintás esemény kezelő.

**onCreateViewHolder és onBindViewHolder metódusok**

```
@NonNull
@Override
public MovieViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
    View view = LayoutInflater.from(context).inflate(R.layout.item_movie,
parent, false);
    return new MovieViewHolder(view);
}

@Override
public void onBindViewHolder(@NonNull MovieViewHolder holder, int position) {
    Movie movie = movies.get(position);
    String baseUrl = "https://image.tmdb.org/t/p/original";
    Glide.with(context)
            .load(baseUrl + movie.posterPath)
            .centerCrop()
            .into(holder.posterImageView);

    holder.itemView.setOnClickListener(v ->
listener.onItemClick(holder.getAdapterPosition()));
}

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

- **onCreateViewHolder**: Nézet létrehozása az egyes listaelemekhez.
- **onBindViewHolder**: Nézet adatokkal való feltöltése (Glide használata a képek betöltésére).

ViewHolder osztály

```
public static class MovieViewHolder extends RecyclerView.ViewHolder {
    ImageView posterImageView;

    public MovieViewHolder(@NonNull View itemView) {
        super(itemView);
        posterImageView = itemView.findViewById(R.id.posterImageView);
    }
}
```

## MovieResponse és Movies osztályok:

- **MovieResponse**: Ez az osztály a JSON válasz adatstruktúráját reprezentálja, amelyet minden API hívásnál kapunk. Egy oldalon (page) 20 filmet tartalmaz, és tárolja az oldalszámot, az összes elemet, valamint a filmeket tartalmazó listát.
- **Movies**: Ez az osztály egyetlen film adatait tárolja, és a **MovieResponse** osztályban listaként van jelen, hogy beleférjen a 20 film.

## MovieDao, AppDatabase, és Movie osztályok:

- **MovieDao**: Ez az interfész tartalmazza az adatbázisműveleteket különböző függvények formájában, például beszúrás, törlés és lekérdezés.
- **AppDatabase**: Ez az absztrakt osztály hozza létre és konfigurálja a helyi adatbázist a Room segítségével. Tartalmazza a DAO (Data Access Object) példányát is.
- **Movie**: Ez az osztály az adatbázis entitásokat reprezentálja, és a filmek adatait tárolja.