One Day Project

영화 정보 서비스

김소정

메인 3초간 나오는 스플래쉬이미지
로고등의 다양한 정보를 담을 수 있다

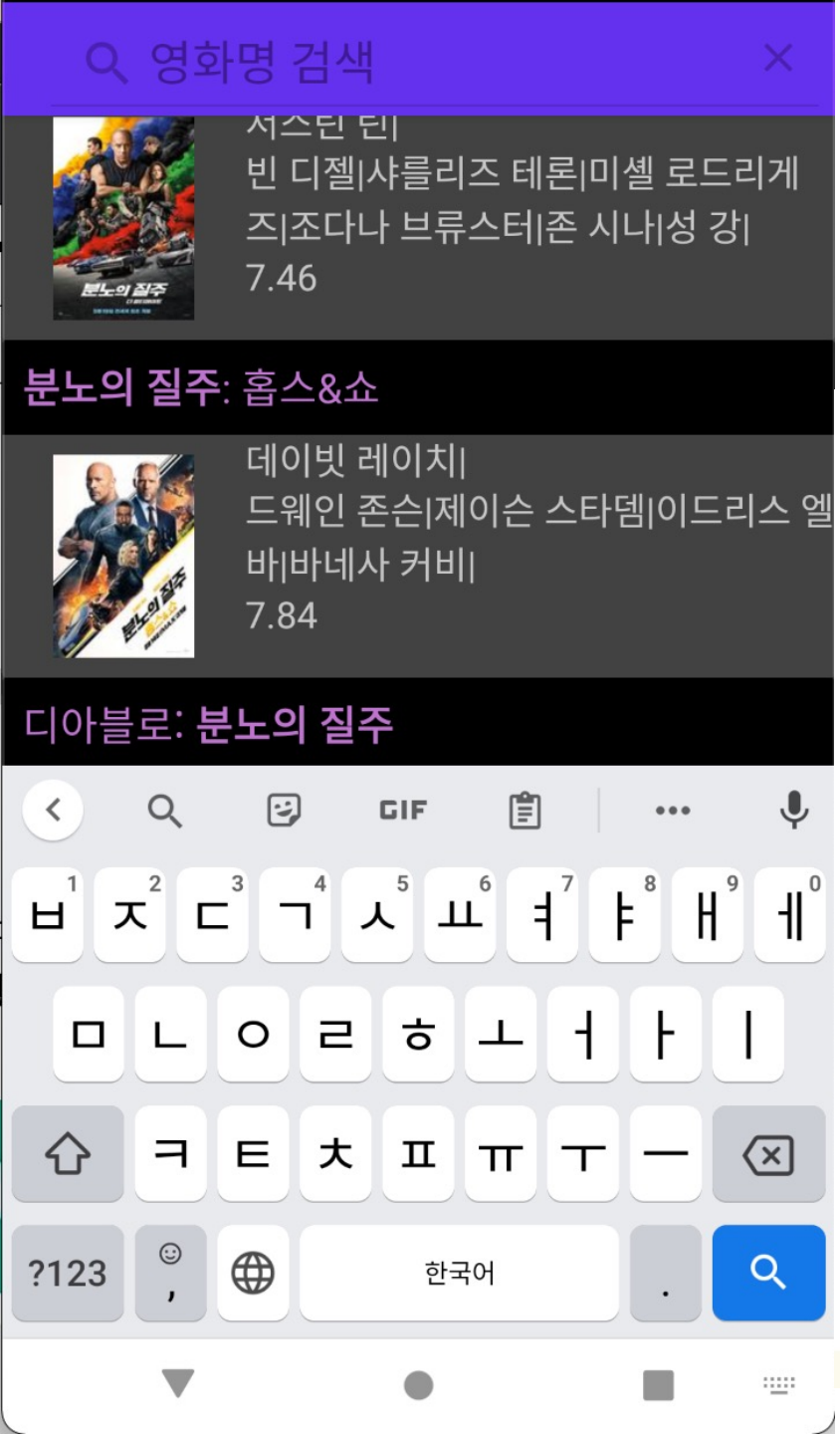
```
public class Splash extends Activity {
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.splash);

        Handler handler = new Handler();
        handler.postDelayed(new splashhandler(), delayMillis: 3000);

    }

    private class splashhandler implements  Runnable{
        public void run(){
            startActivity(new Intent(getApplication(), MainActivity.class));
            Splash.this.finish();
        }
    }
```

Naver Open API의 영화 검색 서비스를 사용하여
안드로이드 스마트폰 어플을 구현합니다




```java
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Toolbar main_toolbar = findViewById(R.id.main_app_toolbar);
    setSupportActionBar(main_toolbar);



    recyclerView = findViewById(R.id.movie_list_view);

    NaverMovieService naverMovieService = new NaverMovieServiceImplV1(recyclerVi

    naverMovieService.getMovie( search: "분노의 질주");
}
```

검색어에서 원하는 검색어를 입력하면

네이버에서 제공하는 검색결과와 동일하게 정보제공이 가능하다

```
public interface NaverMovieService {
    public NaverMovieDTO getMovie(String search);
}
```

```
public boolean onCreateOptionsMenu(Menu menu) {

    getMenuInflater().inflate(R.menu.main_toolbar_menu,menu);
    SearchView searchView = (SearchView) menu.findItem(R.id.app_bar_search).getActionView();

    searchView.setMaxWidth(Integer.MAX_VALUE);
    searchView.setQueryHint("영화명 검색");

    searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener(){

        @Override
        public boolean onQueryTextSubmit(String query) {

            Log.d( tag: "검색버튼 클릭 : ", query);
            NaverMovieService naverMovieService = new NaverMovieServiceImplV1(recyclerView);
            naverMovieService.getMovie(query.trim());

            return false;
```

기획의도

원하는 정보를 간략하게 어플에 구현 또는 제작함으로써 안드로이드 어플 제작에 대한 전반적인 사용법을 익힘

네이버 API뿐만 아니라 어떤 정보든 API를 이용하여 넓은 범위의 정보를 사용할 수 있도록 하고자함

눈에 띄지않는 심플한 디자인으로 기본적인 컨셉을 맞춤

Retrofit 라이브러리를 이용한 API 연동 (1)

```
public interface NaverRetrofit {

    public Call getMovie();

    @GET("movie.json")
    public Call<NaverParentDTO> getMovie(
            @Header("X-Naver-Client-Id") String ClientId,
            @Header("X-Naver-Client-Secret") String ClientSecret,
            @Query("query") String query,
            @Query("start") int start,
            @Query("display") int dispaly


    );
}
```

Retrofit 라이브러리를 이용한 API 연동 (2)

```
public class RetrofitAPIClient {

    public static NaverRetrofit getApiClient(){
        NaverRetrofit naverRetrofit =  getConnection().create(NaverRetrofit.class);
           return naverRetrofit;
    }
    private static Retrofit getConnection() {
        Retrofit retrofit
                = new Retrofit.Builder()
                .baseUrl(NaverSecret.NAVER_BASE_URL)
                .addConverterFactory(GsonConverterFactory.create())
                .build();
        return retrofit;
    }
```

받아온 API정보를 리스트로 만듦

```
public class NaverMovieServiceImplV1 implements NaverMovieService {
    private RecyclerView recyclerView;

    public NaverMovieServiceImplV1(RecyclerView recyclerView) { this.recyclerView  = recyclerView; }

    @Override
    public NaverMovieDTO getMovie(String search) {
        Log.d( tag: "Naver Service", msg: "Start");

        Call<NaverParentDTO> retrofitReturn = RetrofitAPIClient.getApiClient().getMovie(
                NaverSecret.CLIENT_ID,
                NaverSecret.CLIENT_SECTRET,
                search,
                start: 1,
                dispaly: 10);
```

서버에 요청하고 / 응답받는 코드를 확인함

```
retrofitReturn.enqueue(new Callback<NaverParentDTO>() {
    @Override
    public void onResponse(Call<NaverParentDTO> call, Response<NaverParentDTO> response) {
        int resCode = response.code();
        if(resCode < 300){
            Log.d( tag: "네이버 응답 데이터 : ",
                    response.body().toString());
             dispaly: 10);
retrofitReturn.enqueue(new Callback<NaverParentDTO>() {
    @Override
    public void onResponse(Call<NaverParentDTO> call, Response<NaverParentDTO> response) {...}

    @Override
    public void onFailure(Call<NaverParentDTO> call, Throwable t) {

    }
});

return null;
}
```

서버에 요청하고 / 응답받는 코드를 확인함 (2)

```
            dispaly: 10);
    retrofitReturn.enqueue(new Callback<NaverParentDTO>() {
        @Override
        public void onResponse(Call<NaverParentDTO> call, Response<NaverParentDTO> response) {...}

        @Override
        public void onFailure(Call<NaverParentDTO> call, Throwable t) {

        }
    });

    return null;
}
```

받아온 정보를 XML에 표시하는 Adapter

```
private List<NaverMovieDTO> movieList;

public MovieViewAdapter(List<NaverMovieDTO> movieList) { this.movieList = movieList; }

@NonNull
@Override
public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    LayoutInflater layoutInflater = LayoutInflater.from(parent.getContext());

    View view = layoutInflater.inflate(R.layout.movie_item_view,parent, attachToRoot: false);
    MovieItemHolder viewHolder = new MovieItemHolder(view);

    return viewHolder;
}
```

받아온 정보를 XML에 표시하는 Adapter

```
@NonNull
@Override
public void onBindViewHolder(@NonNull RecyclerView.ViewHolder holder, int position) {

    MovieItemHolder movieItemHolder = (MovieItemHolder) holder;

    NaverMovieDTO movieDTO = movieList.get(position);

    String item_title = movieDTO.getTitle();
    Spanned sp_title = Html.fromHtml(item_title,Html.FROM_HTML_MODE_LEGACY);
    movieItemHolder.movie_item_title.setText(sp_title);



    String item_director = movieDTO.getDirector();
    Spanned sp_director = Html.fromHtml(item_director,Html.FROM_HTML_MODE_LEGACY);
    return viewHolder;
}
```

받아온 정보를 XML에 표시하는 Adapter

```
@NonNull
@Override

    String item_actor = movieDTO.getActor();
    Spanned sp_actor = Html.fromHtml(item_actor,Html.FROM_HTML_MODE_LEGACY);
    movieItemHolder.movie_item_actor.setText(sp_actor);

    String item_userpoint = movieDTO.getUserRating();
    Spanned sp_userrating = Html.fromHtml(item_userpoint,Html.FROM_HTML_MODE_LEGACY);
    movieItemHolder.movie_item_userpoint.setText(item_userpoint);

    if(!movieDTO.getImage().isEmpty()){
        Picasso.get().load(movieDTO.getImage()).into(movieItemHolder.movie_item_image);
    }
}
   Spanned sp_director = Html.fromHtml(item_director,Html.FROM_HTML_MODE_LEGACY);
```

받아온 정보의 리스트형을 크기만큼 보여줌

```
@Override
public int getItemCount() {
        return movieList == null ? 0 :movieList.size();
}
```

정보를 표시할 XML을 변수로 불러오는 과정

```
public static class MovieItemHolder extends RecyclerView.ViewHolder{
    public TextView movie_item_title;
    public ImageView movie_item_image;
    public TextView movie_item_director;
    public TextView movie_item_actor;
    public TextView movie_item_userpoint;

    public MovieItemHolder(@NonNull View itemView){
        super(itemView);
        movie_item_title = itemView.findViewById(R.id.movie_item_title);
        movie_item_image = itemView.findViewById(R.id.movie_item_image);
        movie_item_director = itemView.findViewById(R.id.movie_item_director);
        movie_item_actor = itemView.findViewById(R.id.movie_item_actor);
        movie_item_userpoint = itemView.findViewById(R.id.movie_item_userpoint);

    }
}
```

Naver에 요청할 정보와 값들

```
public class NaverSecret {
    public static final String CLIENT_ID = "H3GYSHhb_p5HCGJ2km_S";
    public static final String CLIENT_SECTRET = "[redacted]";

    public static final String NAVER_BOOK_URL = "https://openapi.naver.com/v1/search/movie.json";
    public static final String NAVER_BASE_URL = "https://openapi.naver.com/v1/search/";

}
```

Naver에 요청할 정보와 값들

```
public class NaverMovieDTO {

    private String link;       //   string  검색 결과 영화의 하이퍼텍스트 link를 나타낸다.
    private String image;      //   string  검색 결과 영화의 썸네일 이미지의 URL이다. 이미지가 있는 경우만 나타난다.
    private String subtitle;   //   string  검색 결과 영화의 영문 제목이다.
    private String pubDate;    //   date    검색 결과 영화의 제작년도이다.
    private String director;   //   string  검색 결과 영화의 감독이다.
    private String actor;      //   string  검색 결과 영화의 출연 배우이다.
    private String userRating; //   integer 검색 결과 영화에 대한 유저들의 평점이다.
```

Naver에 요청할 정보와 값들

```
public class NaverParentDTO {

    public String rss;      //  -   디버그를 쉽게 하고 RSS 리더기만으로 이용할 수 있게 하기 위해 만든 RSS 포맷의 컨테이너이며
    public String channel;  //  -   검색 결과를 포함하는 컨테이너이다. 이 안에 있는 title, link, description 등의 항목
    public String lastBuildDate;//  datetime    검색 결과를 생성한 시간이다.
    //item/
    public List<com.kimbyulook.navermovie.model.NaverMovieDTO> items;// -   XML 포멧에서는 item 태그로, JSON

    public String query;    //  string (필수)    Y   -   검색을 원하는 질의. UTF-8 인코딩이다.
    public String display;  //  integer N   기본값 10, 최대 100     검색 결과 출력 건수를 지정한다. 최대 100까지 가능하
    public String start;    //  integer N   기본값 1, 최대 1000     검색의 시작 위치를 지정할 수 있다. 최대 1000까지 가
    public String genre;    //  string  N   -   검색을 원하는 장르 코드를 의미한다. 영화 코드는 다음과 같다.
                                        //                  1: 드라마 2: 판타지
```