

Mobilne Interfejsy Multimedialne

Rozrywka Firebase



**Silesian
University
of Technology**

Jakub Karmański

rok 3 semestr 6 - letni
Wydział Matematyki Stosowanej
kierunek Informatyka

Spis treści:

Użyte narzędzia i źródła	3
Problemy w projekcie	4
Zaplanowanie widoku aplikacji	4
Rozpoznanie danych przychodzących z używanego API	5
Najważniejsze punkty projektu	6
Podsumowanie	7

Użyte narzędzia i źródła

W projekcie “Rozrywka Firebase” wykorzystane zostały darmowe narzędzia:

1. Android Studio,
2. GIMP,
3. GIT

oraz strony:

<https://design.google/>

<https://developer.android.com/guide/topics/ui/look-and-feel>

<https://www.interaction-design.org/literature/topics/material-design>

<https://stackoverflow.com/>

<https://grafmag.pl/artykuly/czym-jest-material-design-teoria-zasady-materialy-i-przyklady>

<https://www.online-convert.com/>

<https://material.io/design/>

<http://www.omdbapi.com/>

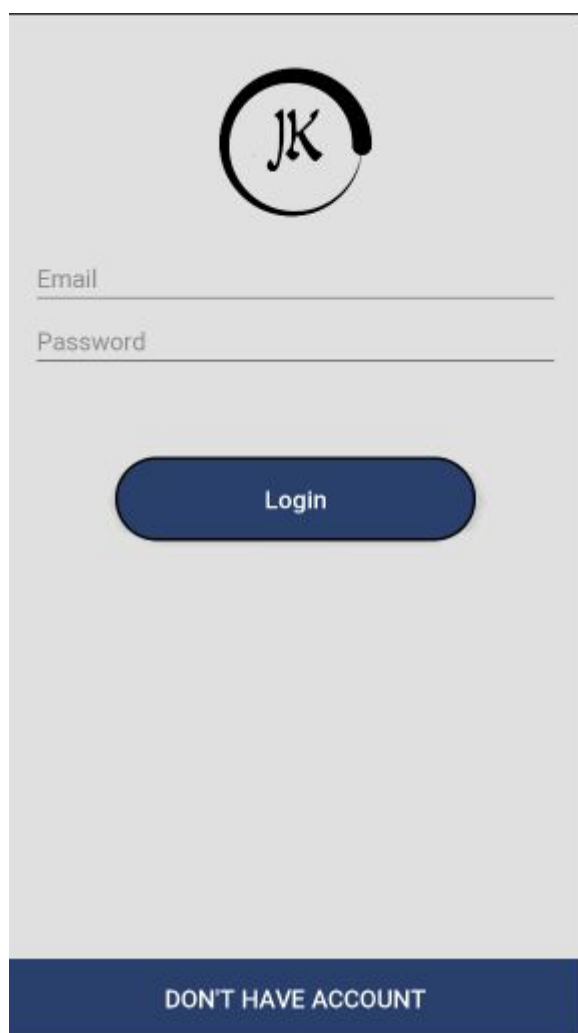
<https://angrytools.com/android/button/>

Problemy w projekcie

Zaplanowanie widoku aplikacji

Był to jeden z najważniejszych problemów projektowych. Problem ten wymaga odpowiedniego rozplanowania dla przejrzystości i ułatwienia korzystania z aplikacji dla jej użytkowników.

Layout musiał być prosty, przejrzysty a zarazem bogaty w informacje potrzebne użytkownikowi.



A mockup of a login form. At the top center is a circular logo containing the letters 'JK' with a stylized headset icon. Below the logo are two input fields: 'Email' and 'Password', each with a horizontal line underneath. Centered below these fields is a dark blue rounded rectangular button with the text 'Login' in white. At the bottom of the form is a dark blue horizontal bar with the text 'DON'T HAVE ACCOUNT' in white.

Rozpoznanie danych przychodzących z używanego API

Problem ten sprowadza się tak naprawdę do szczegółowego zapoznania się ze wszelkimi danymi przychodzącymi z API oraz możliwościami udostępnionymi przez udostępniony punkt dostępu do endpointa - wszelkie zmiany np. języka otrzymywanych danych , formatu , oraz innych kluczowych parametrów endpointu.

Rozpoznanie możliwości dostarczanych z usługi Firebase

Rozpoznanie możliwości pod kątem działania w wybranej technologii oraz języku jest kluczowe dla podejścia do możliwości wykorzystania usługi w aplikacji.

Najważniejsze punkty projektu

Klasa LoginActivity, w której przeprowadzane jest autentykacja użytkownika po wpisaniu maila w prawidłowym formacie oraz hasła.

```
fun login() {
    if (tv_username.text.toString().trim().isEmpty()) {
        tv_username.error = "Please enter email"
        tv_username.requestFocus()
        return
    }

    if (!Patterns.EMAIL_ADDRESS.matcher(tv_username.text.toString()).matches()) {
        tv_username.error = "Please enter valid email"
        tv_username.requestFocus()
        return
    }

    if (tv_password.text.toString().trim().isEmpty()) {
        tv_password.error = "Please enter password"
        tv_password.requestFocus()
        return
    }

    auth.signInWithEmailAndPassword(
        tv_username.text.trim().toString(),
        tv_password.text.trim().toString()
    )

    .addOnCompleteListener(this) { task ->
        if (task.isSuccessful) {
            val user = auth.currentUser
            updateUI(user)
            val intent = Intent(packageContext, MainActivity::class.java)
            intent.putExtra(name="Username", tv_username.text.trim().toString().replace(" ", "").replace(".", ""))
            startActivity(intent)
            finish()
        } else {
            Toast.makeText(
                BaseContext, text="Authentication failed.",
                Toast.LENGTH_SHORT
            ).show()
            updateUI(currentUser: null)
        }
    }
}
```

MainActivity gdzie ładowane są dane o istniejących filmach w bazie danych użytkownika oraz użytkownik ma zapewnioną funkcjonalność usuwania filmu ze swojej kolekcji, czy też dodania filmu do już oglądanych filmów.

```
val listView = findViewById<ListView>(R.id.listViewMyFilms)
listView.setOnItemClickListener { parent, view, position, id ->
    val popup = PopupMenu(context this, view)
    popup.inflate(R.menu.my_film_menu)
    popup.setOnMenuItemClickListener { MenuItem?
        val item = parent.getItemAtPosition(position) as FilmModel
        if (it.title == "Delete") {
            deleteFilm(item.imdbID)
        } else {
            setWatched(item.imdbID, item)
        }
        true
    }
    popup.show()
}
```

```

private fun setWatched(imdbID: String, item: FilmModel) {
    ref.child(imdbID).removeValue()
    item.watched = "FILM ALREADY WATCHED"
    ref.child(imdbID).setValue(item)
}

private fun deleteFilm(key: String) {
    ref.child(key).removeValue()
}

```

FilmService to interfejs, w którym znajduje się serwis zawierający requesty potrzebne do wykonywania operacji na wybranym przeze mnie API.

```

interface FilmService {

    @GET( value: "?")
    fun searchFilm(@Query( value: "s") name: String,
        @Query( value: "apikey") apikey: String): Observable<FilmSearchApiModel.Result>

    @GET( value: "?")
    fun findFilmByImdbID(@Query( value: "i") name: String,
        @Query( value: "apikey") apikey: String): Observable<FilmApiModel.Result>

    companion object {
        fun create(): FilmService {
            val retrofit = Retrofit.Builder()
                .addCallAdapterFactory(RxJava2CallAdapterFactory.create())
                .addConverterFactory(GsonConverterFactory.create())
                .baseUrl( baseUrl: "http://www.omdbapi.com/")
                .build()

            return retrofit.create(FilmService::class.java)
        }
    }
}

```

SearchFilm - w klasie tej znajduje się funkcja wyszukiwania filmów w API oraz dodawania ich do bazy danych użytkownika zalogowanego.

```
private fun beginSearch(searchString: String) {
    var listView = findViewById<ListView>(R.id.listViewSearchFilms)
    var list = mutableListOf<FilmSearchModel>()

    disposable = filmService.searchFilm(searchString, apiKey: "38317b82")
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(
            { result ->
                if (result.Search != null) {
                    var filmList: List<FilmSearchApiModel.Films> = result.Search
                    for (f in filmList) {
                        list.add(FilmSearchModel(f.Title, f.Year, f.imdbID, f.Poster, f.Type))
                    }
                    listView.adapter = SearchFilmAdapter(mContext, this, R.layout.search_films_row, list)
                } else {
                    Toast.makeText(
                        context, this,
                        text: "No results",
                        Toast.LENGTH_LONG
                    ).show()
                }
            },
            { error ->
                Toast.makeText(
                    context, this,
                    error.message,
                    Toast.LENGTH_LONG
                ).show()
            }
        )
}
```

Obiekt FilmSearchApiModel - model wykorzystywany do pobierania danych z API.

```
object FilmSearchApiModel {
    data class Result(val Search: List<Films>, val totalResults: String, val Response: String)
    data class Films(
        val Title: String,
        val Year: String,
        val imdbID: String,
        val Type: String,
        val Poster: String
    )
}
```


Obiekt FilmApiModel - model wykorzystywany do pobierania danych z API dotyczący szczegółowych informacji danego filmu szukanego na podstawie unikalnego id.

```
object FilmApiModel {  
  data class Result(  
    val imdbID: String,  
    val Title: String,  
    val Year: String,  
    val Rated: String,  
    val Released: String,  
    val Runtime: String,  
    val Genre: String,  
    val Director: String,  
    val Actors: String,  
    val Plot: String,  
    val Language: String,  
    val Country: String,  
    val Awards: String,  
    val Poster: String,  
    val Type: String  
  )  
}
```

Podsumowanie

Projekt wymagał odpowiedniej wiedzy na temat API oraz Material Design oraz Firebase. Odpowiednie rozplanowanie widoku danych oraz przenoszenie ich pomiędzy interfejsem serwisu oraz główną aktywnością jest kluczowe dla odpowiedniego efektu aplikacji.

Dzięki prawidłowemu rozplanowaniu danych na widoku oczekiwany efekt jest zadowalający a co najważniejsze funkcjonalny nie tylko w celach edukacyjnych ale także z możliwością wykorzystania dla własnego użytku.