Universidade Federal de Pernambuco – UFPE

Centro de Informática – CIn

Programação para dispositivos móveis

Series Keep

Emanuel Victor - (evfgs)

Edymir Etienne - (eebls)

Recife – 2018.2

Proposta do projeto

Um aplicativo onde o usuário pode marcar quais séries de televisão está acompanhando no momento(episódios e temporadas) e receber notificações sobre lançamentos de novos episódios e notícias sobre estas séries. O usuário também pode marcar filmes que tem interesse e serão lançados nos próximos meses e receber notícias

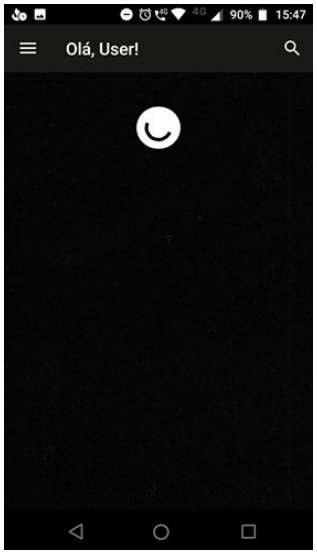
sobre os mesmos.

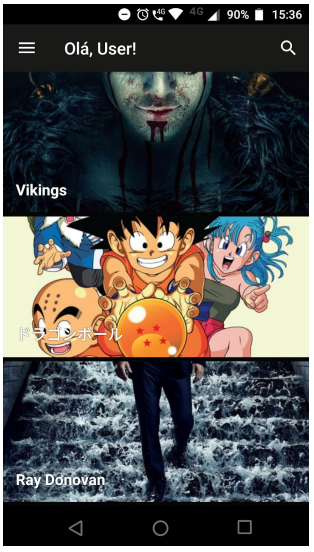
Foi utilizada a API (https://www.themoviedb.org/) que retorna resultados como: original_name, popularity, origin_country, vote_count, first_air_date, overview, poster_path...

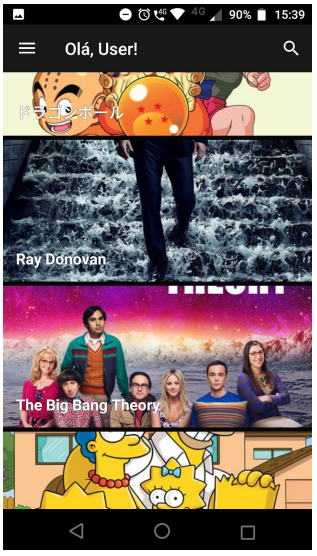
Estrutura do aplicativo

HomePage:

A HomePage é a tela inicial, assim que é inicializada começa a fazer requisição para a API onde vai pegar informação das séries mais populares no momento. Enquanto está fazendo a requisição é mostrada uma barra de progresso circular. Após o retorno e processamento das informações é disposta, em uma lista, as 20 séries mais populares. Contendo o pôster e o título de cada série. Apresenta um Menu e uma opção para buscas

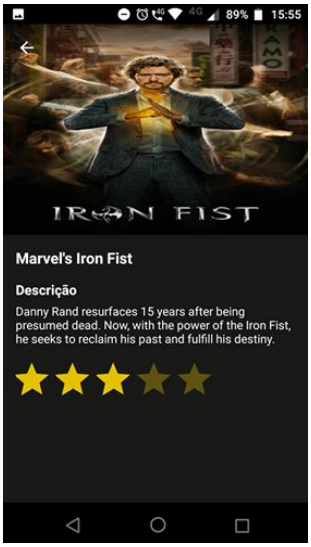


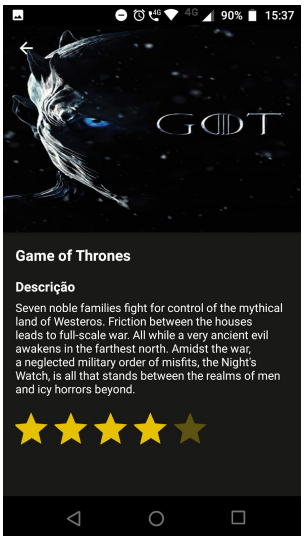




Detalhes das Séries:

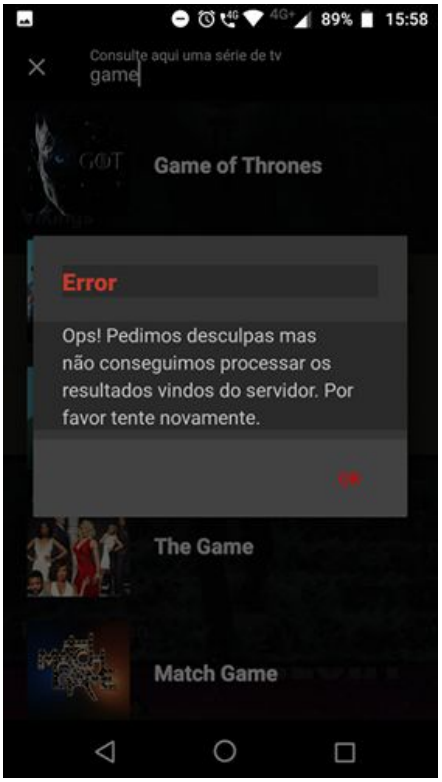
A tela de detalhes das séries mostra o poster completo, a descrição e uma ratingBar que mostra a avaliação dos usuários para aquela série. A página de detalhes apresenta um botão para voltar para a HomePae





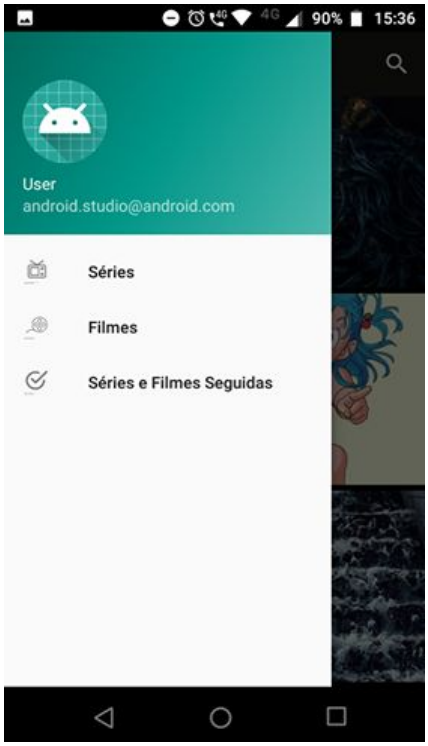
Tela de Buscas:

A tela de buscas apresenta resultados à medida que o usuário vai digitando, porém existe uma limitação que acreditamos que seja da própria API. O ideal teria sido um botão para o usuário acionar quando quiser pesquisar pelo texto digitado. A tela apresenta nos resultados uma imagem menor do pôster e o título da série, ao clicar em um dos resultados o usuário é levado para a página de detalhes da respectiva série



Menu:

Não conseguimos avançar muito no projeto, portanto a tela de menu não encontra-se funcional. Ao clicar num item o usuário é levado para a HomePage, independente do item escolhido. Existe também a opção de acessar o menu ‘puxando’ da lateral da tela a partir da HomePage



Como dito, não conseguimos avançar muito no projeto, portanto o que está funcional é apenas a busca de informações por títulos de séries. Ficamos devendo a parte de notificações de lançamentos de episódios, junto com a parte de notícias, e a integração com os filmes também

Estrutura do projeto

Foram utilizadas as seguintes tecnologias:

Gson:

O Gson é uma biblioteca Java que pode ser usada para converter objetos Java em sua representação JSON. Também pode ser usado para converter uma string JSON em um objeto Java equivalente. O Gson pode trabalhar com objetos Java arbitrários, incluindo objetos pré-existentes dos quais você não possui código-fonte.

```
data class Serie_Instance(
        //Valores de cada serie. Vindos da API
        @SerializedName( value: "poster_path")
        @Expose
        var posterPath: String? = null,
        @SerializedName( value: "overview")
        @Expose
        var overview: String? = null,
        @SerializedName( value: "first_air_date")
        @Expose
        var releaseDate: Date? = null,
        @SerializedName( value: "genre_ids")
        @Expose
        var genreIds: List<Long> = ArrayList(),
        @SerializedName( value: "id")
        @Expose
        var id: Long? = null,
        @SerializedName( value: "original_name")
        @Expose
```

```
    var popularity: Double? = null,
    @SerializedName( value: "vote_count")
    @Expose
    var voteCount: Long? = null,
    @SerializedName( value: "video")
    @Expose
    var video: Boolean? = null,
    @SerializedName( value: "vote_average")
    @Expose
    var voteAverage: Double = 0.toDouble()) : Serializable {
val coverUrl: String
    get() = "${"https://image.tmdb.org/t/p/w500/"}//${posterPath}"

override fun toString(): String {
    return Gson().toJson( src: this)
}
```

Anko:

Utilizada para rodar tarefas assíncronas para carregar a lista de séries e não deixar o usuário travado numa tela enquanto a aplicação espera o processamento da requisição

```
private fun loadSeries(results: List<Serie_Instance>) {
    doAsync { this: AnkoAsyncContext<Homepage_Activity>
        val adapter = AdapterSerie(
                mActivity: this@Homepage_Activity,
                results,
                R.layout.rsc_util_serie_large
        )

        runOnUiThread {
            series.initListItems(
                    activity: this@Homepage_Activity,
                    LinearLayoutManager.VERTICAL,
                    onClickListener: null,
                    adapter as RecyclerView.Adapter<RecyclerView.ViewHolder>
            )
        }
    }
}
```

Dagger 2:

É uma abordagem em tempo de compilação para injeção de dependências

```
@Singleton
@Component(
        modules = {
                MainModule.class,
        })
public interface MainComponent {

    void inject(Homepage_Activity homepageActivity);

    void inject(SerieDetails_Activity activitySerie);

    void inject(SearchSerie_Activity searchSerieActivity);

    void inject(AdapterSerie adapterSerie);
}
```

Retrofit:

Para lidar com a parte de requisições HTTP foi consumida uma API REST utilizando Retrofit, pois a parte de configuração é bem mais simples(configuramos apenas o necessário para estabelecer a comunicação), a questão de serialização dos objetos(não precisamos fazer essa tarefa manualmente) e a usabilidade(é bem mais fácil realizar requisições com o Retrofit).

```
init {
    client =  OkHttpClient.Builder().addInterceptor { chain ->
        var request = chain.request()
        val url = request
                .url()
                .newBuilder()
                .addQueryParameter(
                        "api_key",
                        "3e910d33b3410c4a033c82e9dbc2ae52"
                ).build()

        log( content: "Performing HTTP request to: $url")

        request = request.newBuilder().url(url).build();
         ^addInterceptor chain.proceed(request)
    }.build()

    gson = GsonBuilder()
            .setDateFormat("yyyy-MM-dd")
            .create()

    retrofit = Retrofit.Builder()
            .baseUrl("https://api.themoviedb.org/3/")
            .addConverterFactory(GsonConverterFactory.create(gson))
            .client(client)
            .build()
```

```
interface RestAPIEndPoint {

    @GET( value: "tv/popular/")
    fun getSeries(): Call<ListSeriesResponseWrapper>

    @GET( value: "search/tv/")
    fun searchSeries(@Query( value: "query") name: String): Call<SearchSeriesResponseWrapper>
}
```

Picasso:

Utilizado para baixar, processar e configurar as imagens dentro da aplicação

```
private fun loadSerieDetails() {
    loadCover()
    serie_title.text = mSerieInstance!!.title
    rating_bar.rating = (mSerieInstance!!.voteAverage * 5 / 10).toFloat()
}

private fun loadCover() {
    Picasso.with( context: this).load(mSerieInstance!!.coverUrl).fit().into(cover)
}
```