- Dividir todo o projeto em pastas, assim ficando melhor a organização do projeto.

- Seguir sempre um padrão para se organizar, e utilizar um domínio para não se perder.

- Layout é feito com um código XML.

- Para utilizar os resources, utilizasse a classe R.

- Toda view ou viewgroup deve ter dois atributos, layout read e layout , se não tiver os dois, vai dar erro.

- O tamanho de pixel é diferente de um celular para outro, pois os tamanhos da tela variam.

- Sempre pensar na responsividade do aplicativo.

- Existem parametros no activy_main que definimos os tamanhos, sendo eles, layout_width, layout_height.

- Orientation mostra se os filhos serão colocados na horizontal ou vertical. Podendo ser utilizado os pixel fixos ou o match_parent, que acompanha o tamanho do pai ou o wrap_content que acompanha o tamanho do filho.

- Existe também o padding e a margin, que serve para dar um "respiro" no layout e deixar ele mais elegante.

- Para utilizar o margin, deve se usar start e end ao invés de utilizar left e right.

- Para puxar o valor de qualquer outro resource basta usar o @nomedoresource.

- Para um bom projeto, é necessário utilizar os values, para pré-definir padrões de cores, dimensões e strings, isso ajuda na hora de criar e orgnaizar qualquer tela.

- Utilizar espaçamento sempre, para a aplicação ficar mais organizada. Não deixar muita informação agrupada.

- Utilizar TextInputLayout como caixa de texto. Sendo ele mais moderno.

- imeOption é utilizado para definir como o teclado irá se comportar, exemplo: ao clicar no enter, pule linha ou até mesmo, ao clicar em Enter ele envie o formulário por exemplo. Existem diversos modelos para utilizar no imeOption.

- inputType podemos personalizar por exemplo na hora de formatar o campo, por exemplo utilizar email, deixar a primeira letra maiscula, configurar para que sugira os e-mails já utilizados naquele campo, ou deixar o campo apenas para digitar números.

- LinearLayout é utilizado para criar um campo em linha.

- ScrollView cria um scroll, sendo ele Vertical ou Horizontal.

- Card é utilizado para deixar mais bonito e organizado a aplicação, é muito recomendado a ser utilizado

- Utilizar sempre sp e dp no texto para acompanhar o modo acessibilidade do celular.

- FrameOut vai empilhar os filhos.

- Utilizar LinearLayout para empilhar.

- Para acessar um atributo, usar sempre o FindByID

- Toda imagem deve ter um nome, aceita a maior parte de formatos de imagem, e a partir do Android 8 pode ser utilizado SVG.

- Toda imagem deve ter uma restrição na horizontal e vertical.

- A imagem tem que ser ancorada a um pai para não acabar saindo da tela.

- Montar sempre o esboço do aplicativo para depois começar a construir dentro do Aplicativo.

- Para melhor organização, começar sempre a montar o design do topo esquerdo para direito e parte inferior.

- Para alinhar tudo, são necessários fazer as restrições, para que os componentes não fiquem fora de área e posição.

- É melhor montar tudo dentro de um Card, para quando tiver uma modificação, fazê-la apenas no Card.

- As atividades são cadastradas no androidManifest.

- Para acessar componentes de tela utiliza-se o findViewById, caso seja um botão, utiliza-se o findViewById<Button>(R.id.nomedocomponente)

- Para fazer o DataBinding é necessário utilizar bibliotecas. Elas fazem o trabalho de declaração automatica dos componentes. Isso faz com que economize tempo ao digitar os códigos.

- Databinding deve ser habilitado no gradle, modificar nos arquivos do layout e substituir o chamado do setContentView()

- Databinding além de facilitar em códigos ele ajuda na-Dependencias é o local que devemos colocar todas as bibliotecas que desejamos usar.

- Para publicar a aplicação no play store é sempre necessário alterar a versão do projeto.

- Existem diferentes formas de cadastrar eventos em um componente.

- Via Lambda (usado quando o evento não precisa ser reaproveitado, são poucas linhas de código).

- Via método (usado quando o evento precisa ser reaproveitado, podem ser função complexas)

- Tudo que começa em <on> termina em <listener> é um evento.

- Os eventos são disparados pelos componentes após a interação do usuário.

- Alguns componentes só aceitam um evento, porém existe componentes que aceitam mais, depende do que for passado para ele.

- Fragmento é uma tela que se acopla a uma atividade, dentro de uma atividade é possível acoplar diversos fragmentos.

- Atividade e Fragmento tem ciclos de vida.

- Na criação de fragmento, temos a mudança de status, e ele se interliga a atividade.

- O fragmento não pode ser criado antes da atividade.

- Quando o fragmento está ativo, significa que ele está interagindo

- Fragmento pode ser reaproveitado em outros projetos ou contextos.

- Valor absoluto só deve ser usado em imagem. Nunca em botão e em Constrains.

- É necessário criar uma pasta dentro do res para subdividir e dentro dela criar um resource, como por exemplo a criação de um Menu

- Servidores HTTP (Apache, Tomcat, Nginx, IIS)

- Banco de Dados Relacional (MySql, SQL Server, Postgree)

- Banco de dados noSQL (MongoDB, Firebase)

- Comunicação do servidor de aplicação pode ser junto ou separado.

- A comunicação é feita através da arquitetura Orientada a Serviço

- SOAP, Simple object acess protocol, utiliza XML sobre HTTP para a comunicação

- XML começou a ser usado em 1996, representação de documento semi-estruturado, é composto por elementos e atributos.

- Rest é padrão de mercado, é utilizado para criação de web-service.

- Clientes magros são clientes que compreendem uma unica camada, não apresentam codigo de aplicação personalizado, são totalmente dependentes do servidor e utilizam navegadores web.

- Clientes gordos possuem três camadas, interface com usuário, logica de negócio e acesso de dados. comunicação entre clientes e servidor é baixo. Ex: WhatsApp

Tipos de aplicações: - Nativo, desenvolvido direto na plataforma Android. Auto desempenho, utiliza o ecossistema da plataforma, necessita maior esforço de desempenho

- Compile-to-native, Ambiente de terceiro, aplicação compilada para diversas plataformas, dificuldade no dominio dos framworks, exemplos: React Native, Native Script, Xamann

- Hibrida, fácil para desenvolvedores WEB, executam em uma webview - pode ficar lento, Exemplo: PhoneGap, Cordova, Ionic.

- Progressive Web App, Facil de desenvolver, não sao aplciativos reais, executam no navegador.

- Volley é uma biblioteca para chamadas de API do google, mas não pode ser chamado no thread principal.

- Todo ano é necessário fazer a atualização do SDK.

- Cada API tem sua dinâmica e funciona de forma diferente, e tem seu mapeamento diferente.

- Todos os componentes que estão nas respostas precisam estar na classe.

- jsonschema2pojo é o site utilizado para transformar o json para uma classe.

- Callback recebe uma chamada que vem na resposta.

- Se for usar poucas vezes, podemos gerar uma classe anônima.

- Callback precisa estar na API Service.

- OnResponde se a resposta deu certo

- OnFailed se a resposta deu falha.

- Cada aplicativo tem sua área de trabalho privada, onde todos os arquivos ficam dentro do próprio aplicativo, sendo que pode salvar na área pública caso seja uma opção desejada.

Arquivo local: txt(csv,json,xml), dat (binário)

- Sem internet se torna + rápido.

- É mais fácil.

- Ocupa mais espaço em disco e mais memória.

- Leitura.

- Relacionamento.

- Atualização.

Sqlite: banco de dados

- Suporta maior quantidade de dados.

- Mais facilidade ao localizar o dado.

- Mais fácil inserir, atualizar, deletar.

- Mais lento.

- Relacionamento complexo.

- É necessário mais código.

Shared Preference

- Grava chave + valor.

- Pode se utilizar para gravar configurações locais.

- Quando reinstala ele recria essas informações.

- Muito utilizado para preferencias.

- Pode utilizar também como cache.

- Mais fácil de usar e não precisa configurar nada.

API

- É necessário utilizar internet sempre.

- Suporta maior quantidade de dados

- Utiliza muito pouco espaço do dispositivo

- POST(Create) - GET(Read) - PUT PATCH(Update) - DELETE(Deleta)

========== **Fragmento e carregar dados API** ==========

```
class Tela1Fragment : Fragment() {
  private lateinit var mBinding: FragmentTela01Binding
  private var mFilmes: MutableList<Filmes> = mutableListOf() // Lista com
todos os Filmes
  private var mPagina = 1   // Controle da paginação do webservice
  override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?
  ): View {
    // Inicialização usando DataBinding
    mBinding = FragmentTela01Binding.inflate(inflater, container, false)
    return mBinding.root
  }
  private fun carregarFilmes() {
            Log.d(TAG, "carregarFilmes: ")
            // Verifica se chegou na última página
            if (mPagina < 0) {
            Log.i(TAG, "carregarFilmes: Última página - sem dados para
carregar")
            return
            }
            Log.d(TAG, "carregarFilmes: Carregando dados da página
$mPagina")
              val call = NetworkManager.service.listarFilmes(mPagina)
            // Enfileira a execução do webservice e trata a resposta
            call.enqueue(object : Callback<Response<Filmes>> {
            // Retorno de sucesso
            override fun onResponse(
            call: Call<Response<Filmes>>,
            response: retrofit2.Response<Response<Filmes>>
            ) {
          onResponseSuccess(response.body())
            }
            // Retorno de falha
            override fun onFailure(call: Call<Response<Filmes>>, t:
Throwable) {
            Log.e(TAG, "onFailure: ", t)
            if (context != null) {
            Toast.makeText(context, t.message,
Toast.LENGTH_LONG).show()
            }
            }
            })
            }
  <!— AndroidManiFest - Utilização de internet para chamada aos
webservices -->
  <uses-permission android:name="android.permission.INTERNET"/>
  <!-- Permite que o Glide recarregue as imagens caso ocorra erro na rede
-->
  <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

========== **Layout** ==========

```
// <!-- Variáveis de amarração
<data>
            <variable
            name="filmes"
        type="com.example.ed09_ate_11.models.Filmes" />
            </data>
            <!-- Layout da tela, começo do layout.
  <com.google.android.material.card.MaterialCardView
            <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="@dimen/spacing_normal">
        <de.hdodenhof.circleimageview.CircleImageView
          android:id="@+id/filmes_img_imagem"\\recycleview cria
automático
          android:layout_width="185px"
            android:layout_height="104px"
          app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            tools:src="@tools:sample/avatars" />
```

========== **Base da API** ==========

```
object NetworkManager {
            // URL base da API
            private val URL = "https://list.ly/api/v4/lists/6/"
            lateinit var client: OkHttpClient
            val service: ApiService by lazy {
            val interceptor = HttpLoggingInterceptor()
            interceptor.level = HttpLoggingInterceptor.Level.BODY
            client =
OkHttpClient.Builder().addInterceptor(interceptor).build()
            val retrofit = Retrofit.Builder()
            .baseUrl(URL)
            .client(client)
        .addConverterFactory(GsonConverterFactory.create())
            .build()
            return@lazy retrofit.create(ApiService::class.java)}
            fun stop() {
            client.dispatcher().cancelAll()}
```

========== **Mapeia endpoint da API** ==========

```
interface ApiService {
            @GET("items")
            fun listarFilmes(@Query("page") pagina: Int):
Call<Response<Filmes>>
```

========== **Exemplo de Classe** ==========

```
data class Filmes (
            @SerializedName("id") var id: Int = 0,
            @SerializedName("name") var name: String = "",
            @SerializedName("url") var url: String = "",
            @SerializedName("images") var images: Images? = null,
)
data class Images (
            @SerializedName("small") var small: String = "",
            @SerializedName("large") var large: String = "",
 )
            Glide.with(holder.itemView) // Carrega a imagem do filme
            .load(filme.images?.small)
            .centerCrop()
        .placeholder(R.drawable.ic_placeholder)
        .into(holder.binding.filmesImgImagem);
            Log.d(TAG, "Img Filme: ${filme.images?.small}")
```

========== **Eventos** ==========

```
  -- Definição dos eventos que um item do recycler view pode disparar. A
ação executada
  -- em cada evento será definida pela classe que criou o recycler view.
            interface Evento {
            fun onCompartilharClick(filmes: Filmes)
            fun onFilmesClick(filmes: Filmes)}
```

========== **View Holder** ==========

```
 * Classe do ViewHolder que armazena os itens de layout do recycle view
            data class ViewHolder(var binding: ItemFilmesBinding) :
            RecyclerView.ViewHolder(binding.root)}
```