- Scrollview - Dependendo do tamanho do dispositivo não é recomendado.
- Linear Layout - Principais são layout_width e o layout_height temos três atributos match_parent que é expandir na altura e largura com base no componente pai, wrap_content seria o tamanho do meu conteúdo para os “filhos” e absoluty. Nunca utilizar absoluty para não causar problemas de responsividade.
- ImageView - Não será obrigatório largura e altura. Se atentar para não utilizar espessura de fontes dependendo do contraste das cores.
- TextView – Para deixar texto estático.
- Dimens - utilizado com valores já pré setados para uma grid de uma tela e podem ser criados outros para dispositivos com dimensões diferentes.
- Color - com cores já definidas e pode ser criando arquivos, por exemplo para criar um modo noturno para aplicação.
- Margin – Top, Bottom, Start, End.
- Para utilizar pacotes de terceiros, colocar o caminho inteiro do pacote na hora que for de utilizalo.
- TextInputLayout - Caixa de texto com valor representando para o que o campo se trata.
- TextInputEditText
- imeOptions – Pode definir o que o Enter fará na aplicação como passar para o próximo ou finalizar processo e mundo a representação gráfica.
- InputType – Editar o teclado para puxar dependendo do campo que vai ser preenchido. Como colocar um teclado numérico para inserir um número de telefone.
- Linear-Layout – Pode ser utilizado dentro de outro para criar um campo em linha que inserindo um componente dentro ele se comporta internamente.
- HorizontalScrollView - Criar um Scroll na horizontal. Na maioria das vezes utilizado com o Linear-Layout.
- Para texto utilizar sp e para outros componentes utilizar dp.
- contentDescription – Para criar uma descrição na ausência da informação.
- FremeLayout – Sobrepõem os filhos.
- Conteiner – Agregados aos outros.
- Recurso - @+nome para criá-lo.
- findViewById - Para acessar qualquer atributo criado.
- Adicionar nova biblioteca: colocar o código na buid.gradle(Module)
- Imagens - Qualquer tipo de extensão, porém apenas do android 8 funciona as com (.svg). Para imagens sempre tem que definir uma restrição na vertical ou na horizontal.
- Pode ancorar o componente na interface gráfica, selecionando as restrições. Colocar largura 0dp para ocupar toda a área da restrição.
- Sempre cadastrar as atividades no androidManifest.
- Padrão de nome para colocar no resource seria o título da tela.
- Mudar a cor do btn seria no backgroundtint.
- Quando aparecer a linha em amarelo Alt + Enter para visualizar o erro que está ocorrendo.
- Data binding necessários algumas bibliotecas.
- Configuar databinding - Habilitar o dataBinding no gradle, Modificar os arquivos de Layout, Substituir a chamada do setContentView() - Fazer isso no buildGrade.
- Data binding é uma técnica geral que une duas fontes de dados/informações e as mantém em sincronia em um processo que estabelece uma conexão entre UI da aplicação e a lógica de negócio.
- Lateint para variável nunca ser nula.
- Componente setOnClickListener() para tratamento de eventos, os eventos são disparados pelos componentes após a interação do user.
- Via função Lambda - para eventos curtos e não precisam ser reaproveitados.
- Via método - para eventos curtos e podem ser reaproveitados.

Fragmento

- Proposta, basicamento juntar varias activity
onAttach() Uni o fragmento com a atividade
onCreate() O elemento foi criado
onCreateView() Criar a tela
onActivityCreated() Atividade já foi criada
onStart()
onResume()
Fragment is active - Interagindo com ele
onPause()
onStop()
onDestroyView()
onDestroy()
onDetach()
Fragmento is destroyed
- Valor absoluto apenas para img, quando for um btn utilizar warp_containt
- Para o fragmento
- Criar pasta menu, apos resourceDirector
- Vector para adicionar mais icones
- Material Desing icons para coletar mais icones, apenas copiar o xml e criar um novo arquivo xml dentro da pasta drawable
- ifRoom se tiver espaço ele coloca na tollbar, no caso no exemplo de um menu
- always apenas uma forma
- showAsAction: define como vamos exibir o menu superior
- opp:showAsAction
- Para construir um fragmento, usamos o padrão builder
- Ele permite que todos os fragmentos sejam criados de maneira uniforme, padroniza a passagem de parametros
- Função estática para a construção dos fragmentos
- JvaStatic
- !! forçar uma variavel a ser nula

Banco de Dados

- Banco de Dados Relacional: MySQL, PostgreSQL
- Banco de Dados noSQL: Firebase
- Servidor por comunicação HTTP
- Servidor de aplicação : comunicação TCP/UDP
- Protocolos:
- SOAP usa (XML + HTTP) para comunicação

- XML composto por elementos e atributos
- REST utilizado para criação de web-services
- Cliente fornece interação com o usuario
- Clientes magros:
- Compreendem uma única camada, nâo apresentam código de aplicação personalizado, são totalmente dependentes do servidor
, utilizam navegadores web.
- Clientes gordos:
- Possuem até três camadas, apresentação interface com usuário, nogócios lógica de negócios,
- Aplicações:
- Desenvolvidos diretamente para a plataforma, alto desempenho, utlização de todo ecossistema
, necessita de maior esforço de desenvolvimento.
- compile-to-native: ambiente de terceiros, aplicação compilada para diversas plataformas, dificuldade no domínio dos frameworks,
ex: React Native, Native Script, Flutter, Xamarin.
- Híbridas: Fácil para desenvilvedores web, executa em uma web-view - pode ser lento, Ex: PhoneGap, Cordova, Sencha, Ionic.
- Progressive Web App (PWA): Fácil de desenvolver, não são apicativos reais, executa no navegador, Exemplos: HTML/CSS
Parte Cliente Servidor
- Http: Protoco para troca de mensagens na web.
- Rest: não pode ter uma sessão
- Entidades = URLs: Aluno, Livro, Professor, Produto
- Ações Comandos HTTP: GET, POST, DELETE, PUT
- GET: não altera o estado do servidor
- POST: pode alterar o estado a qualquer momento
- PUT, PATCH e DELETE são idempotente, chamadas repetidas nao importam.
- Boas praticas: Manter nomes no plural, Enviar abstrações, Paginação, Subconjunto de campos
- BLOB:
- RAW: não pode ser combindado com outros tipos, para HTTP se utiliza o MIME para identificação
- URL: resquest separado para acesso
- Encoding Base64: Aumenta o tamanho em 33%
- Códigos de erro:
- 200 - OK, 201 - Criado, 401 - Acesso não altorizado, 404 - Recurso não encontrado, 500 - Erro interno bug
- Autenticação, simples, assinturas de APIs, OAuth 2.0, JWT.

- **API**
- Deve se criar uma thread para as chamadas, pra nao ocasionar o fechamento do sistema.
- Volley para chamada da API.

```
// Lista com todos os Filmes
private var mFilmes: MutableList<Filmes> = mutableListOf()
```

Android Studio
- Todo ano ficar atualizando o sdk para os aplicativos na Play Store.
- Retrofit: Importar as bibliotecas para chamadas webserver.
- Gson: também as para serializar objetos Java em JSON.
- Encapsulado no arquivo de inicialização da API NetworkManager.kt.
- ApiServer.kt: Interface que mapeia.
- Criar classes para os dados da API.
- jsonchema2pojo para criar as classes apenas colando o codigo da API.
Persistências de Dados
Arquivo Local
- txt(.csv, .json, .xml)
- dat(binário)
- Vantagens
-- Sem Internet
-- Velocidade
-- Fácil configuração
- Desvantagens
-- Tamanho disco limitado
-- Leitura
-- Não conseguir fazer relacionamento com outros dados
-- Atualização dos dados
SqLite
- Vantagens
-- Maior volume de dados
-- Consultas customizadas
-- Fácil insert/delete/update
-- Relacionamento complexo
-- Mais tipos de dados
- Desvantagens
-- Pesado
-- Criação de código limitado
-- Atualização de banco
Shared Preferences
- Vantagens
-- Grava chave e valores
-- Preferencia na aplicação
-- Serve como cache
-- De fácil uso
- Desvantagens
-- Volumes de dados limitado
-- Desviando funcionalidade
API (firebase)
- Vantagens
-- Maior armazenamento de dados
- Desvantagens
-- Necessita de internet 100% do tempo

- Comando	Http - Operação BD	- /gatos	- /gatos/3	
- POST:	Create	Cria um gato novo	-	
- GET:	Read	Lista todos os gatos	Lista delalhes do gato 3	
- PUT PATCH:	Update	-	altera todos os detalhes do gato 3	
- DELETE:	Delete	Deleta todos os gatos	Deleta o gato 3	

--Fragment e carregar dados API----------

```
class Tela1Fragment : Fragment() {
   private lateinit var mBinding: FragmentTela01Binding
   // Lista com todos os Filmes
   private var mFilmes: MutableList<Filmes> = mutableListOf()
   // Controle da paginação do webservice
   private var mPagina = 1
   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()
   }---- Interface que mapeia todos os endpoints 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 = "",
 )
// Carrega a imagem do filme---------------------------------------------------------------------------
      Glide.with(holder.itemView)
         .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)
}
```