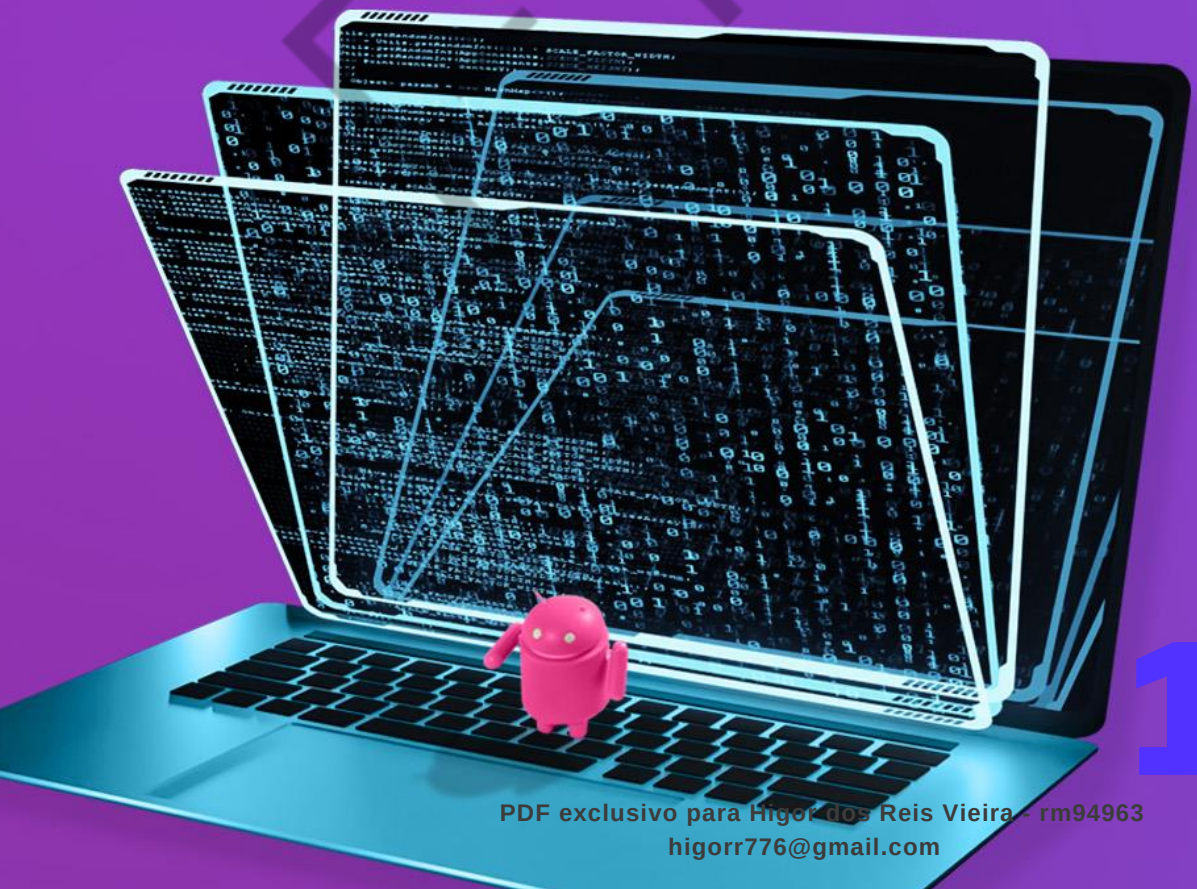
APP WORLD

LISTAS E **CONSUMO DE API EXTERNA**



12A

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 LISTAS E CONSUMO DE API EXTERNA

1.1 LazyColumn

Uma das tarefas mais comuns quando desenvolvemos aplicações Android é exibir informações na forma de lista, como uma lista de produtos, de contatos, de mensagens etc. Anteriormente ao Jetpack Compose utilizávamos um componente chamado “RecyclerView”, que era bastante trabalhoso. Hoje, a construção desses componentes se tornou bastante simples com a utilização dos composables “LazyColumn” e “LazyRow”. Neste momento, vamos explorar a utilização do “LazyColumn”.

Crie um projeto no Android Studio chamado “Listas Lazy” e apague as funções “Greeting” e “GreetingPreview”.

Para o nosso projeto vamos precisar de 2 pacotes: “model” e “repository”. A estrutura do projeto deverá se parecer com a figura “Estrutura inicial do projeto”:

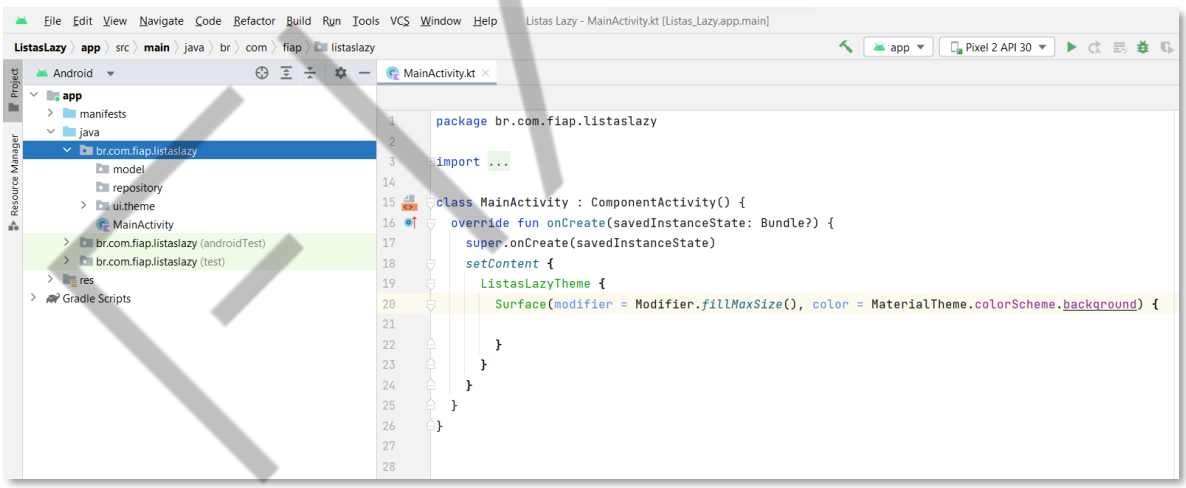


Figura 1 – Estrutura inicial do projeto
Fonte: Elaborado pelo autor (2023)

No pacote “model” vamos criar uma classe de dados chamada “Game”. O conteúdo desta classe pode ser visualizado na listagem “Classe Game”, logo abaixo:

```
package br.com.fiap.listaslazy.model

data class Game(
  val id: Long = 0,
  val title: String = "",
  val studio: String = "",
  val releaseYear: Int = 0
)
```

Código-fonte 1 – Classe Game
Fonte: Elaborado pelo autor (2023)

Agora, no pacote “repository” vamos criar um arquivo chamado “GamesList.kt”. O código deste arquivo está disponível na listagem “Arquivo GamesList.kt”:

```
package br.com.fiap.listaslazy.repository

import br.com.fiap.listaslazy.model.Game

fun getAllGames(): List<Game> {
  return listOf(
    Game(id = 1, title = "Double Dragon", studio = "Technos", releaseYear
= 1987),
    Game(id = 2, title = "Batletoads", studio = "Tradewest", releaseYear
= 1991),
    Game(id = 3, title = "Enduro", studio = "Activision", releaseYear =
1983),
    Game(id = 4, title = "Ikari Warriors", studio = "SNK", releaseYear =
1986),
    Game(id = 5, title = "Captain Commando", studio = "Capcom",
releaseYear = 1991),
    Game(id = 6, title = "Mario Bros", studio = "Nintendo", releaseYear =
1983),
    Game(id = 7, title = "Tiger Heli", studio = "Taito", releaseYear =
1985),
    Game(id = 8, title = "Mega Man", studio = "Capcom", releaseYear =
1987),
    Game(id = 9, title = "Gradius", studio = "Konami", releaseYear =
1985),
    Game(id = 10, title = "Gun Fight", studio = "Taito", releaseYear =
1975)
  )
}

fun getGamesByStudio(studio: String): List<Game>{
  return getAllGames().filter {
    it.studio.startsWith(prefix = studio, ignoreCase = true)
  }
}
```

Código-fonte 2 – Arquivo GamesList.kt
Fonte: Elaborado pelo autor (2023)

O arquivo “GamesList.kt” implementa a função “getAllGames()”, que retorna uma lista com dez games e a função “getGamesByStudio()”, que retorna uma lista de games cujo nome do estúdio começa com o valor do argumento “studio” ignorando maiúsculas e minúsculas.

Vamos criar agora a interface do usuário. O arquivo “MainActivity.kt” deverá se parecer com a listagem “Arquivo MainActivity.kt inicial”:

```
package br.com.fiap.listaslazy

import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Search
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import br.com.fiap.listaslazy.repository.getGamesByStudio
import br.com.fiap.listaslazy.ui.theme.ListasLazyTheme

class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    Log.i("aaa", getGamesByStudio("Capcom").toString())
    setContent {
      ListasLazyTheme {
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {
          GamesScreen()
        }
      }
    }
  }
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun GamesScreen() {
  Column(modifier = Modifier.padding(16.dp)) {
    Text(
      text = "Meus jogos favoritos",
      fontSize = 24.sp,
      fontWeight = FontWeight.Bold
    )
```

```
    Spacer(modifier = Modifier.height(16.dp))
    OutlinedTextField(
      value = "",
      onValueChange = {},
      modifier = Modifier.fillMaxWidth(),
      label = {
        Text(text = "Nome do estúdio")
      },
      trailingIcon = {
        IconButton(onClick = { /*TODO*/ }) {
          Icon(
            imageVector = Icons.Default.Search,
            contentDescription = ""
          )
        }
      }
    )
    Spacer(modifier = Modifier.height(16.dp))
  }
}
```

Código-fonte 3 – Arquivo MainActivity.kt inicial
Fonte: Elaborado pelo autor (2023)

Execute a aplicação em um emulador. O resultado esperado deve se parecer com a figura “Layout inicial da aplicação”:

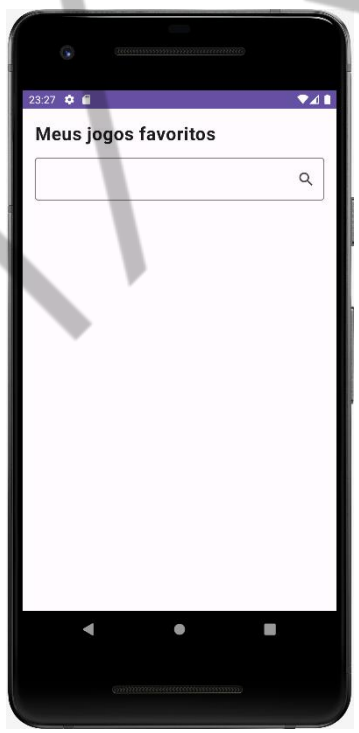


Figura 2 – Layout inicial da aplicação
Fonte: Elaborado pelo autor (2023)

1.1.1 Implementando a LazyColumn

Já vimos que podemos criar listas utilizando “Column” em conjunto com algum laço de repetição, mas isso deve ser feito para exibir listas pequenas, pois, neste caso,

todos os itens da lista serão compostos e colocados no layout estando visíveis ou não, o que pode causar sérios problemas de performance.

Quando trabalhamos com listas muito grandes, a melhor opção é a utilização da “LazyColumn”, que posiciona na lista apenas os itens visíveis, eo restante da lista vai sendo inserido de acordo com a rolagem da lista, garantindo uma performance superior.

Vamos implementar a função “GamesScreen” no arquivo “MainActivity.kt”. A implementação da “LazyColumn” para a nossa lista está disponível na listagem “Função GamesScreen inicial” abaixo:

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun GamesScreen() {
  Column(modifier = Modifier.padding(16.dp)) {
    Text(
      text = "Meus jogos favoritos",
      fontSize = 24.sp,
      fontWeight = FontWeight.Bold
    )
    Spacer(modifier = Modifier.height(16.dp))
    OutlinedTextField(
      value = "",
      onValueChange = {},
      modifier = Modifier.fillMaxWidth(),
      label = {
        Text(text = "Nome do estúdio")
      },
      trailingIcon = {
        IconButton(onClick = { /*TODO*/ }) {
          Icon(
            imageVector = Icons.Default.Search,
            contentDescription = ""
          )
        }
      }
    )
    Spacer(modifier = Modifier.height(16.dp))
    LazyColumn(){
      items(getAllGames()){
        Column() {
          Text(text = it.title)
        }
      }
    }
  }
}
```

Código-fonte 4 – Função GamesScreen inicial
Fonte: Elaborado pelo autor (2023)

Como podemos ver, a implementação da “LazyColumn” é bastante simples. Note que a “LazyColumn” possui a função “items”, que será responsável por

renderizar cada elemento da lista fornecida como argumento. Em nosso exemplo estamos passando a lista de games através da função “getAllGames()”. A cada iteração da lista de games, a função “items” nos devolve um elemento da lista armazenado referenciado pela variável “it”. Por enquanto exibimos apenas os nomes dos games, mas vamos melhorar a exibição. Ao executar o aplicativo, o resultado esperado deverá se parecer com a figura “Exibindo nomes dos games na LazyColumn”:

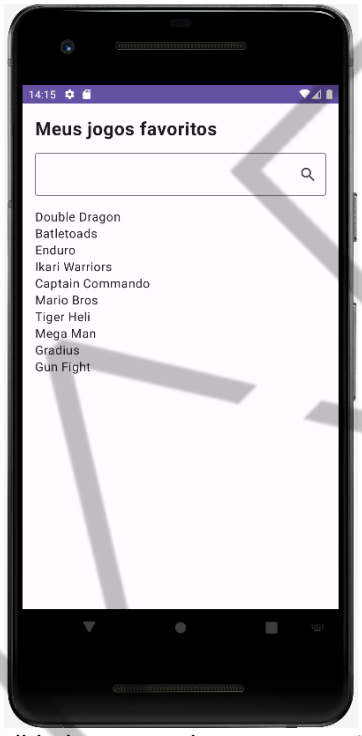


Figura 3 – Exibindo nomes dos games na LazyColumn
Fonte: Elaborado pelo autor (2023)

Vamos melhorar a exibição dos games utilizando um “Card” e adicionando as outras informações. Vamos criar uma função chamada “GameCard” no arquivo “MainActivity.kt” que será responsável por renderizar cada item da lista. A função “GameCard” está disponível na listagem “Função GameCard”:

```
@Composable
fun GameCard(game: Game) {
  Card(modifier = Modifier.padding(bottom = 8.dp)) {
    Row(
      verticalAlignment = Alignment.CenterVertically,
      horizontalArrangement = Arrangement.SpaceBetween,
      modifier = Modifier
      .fillMaxWidth()
    ) {
      Column(modifier = Modifier
        .fillMaxWidth()
```

```
            .padding(16.dp).weight(3f)) {
            Text(
              text = game.title,
              fontSize = 20.sp,
              fontWeight = FontWeight.Bold
            )
            Text(
              text = game.studio,
              fontSize = 14.sp,
              fontWeight = FontWeight.Normal
            )
          }
          Text(
            text = game.releaseYear.toString(),
            modifier = Modifier.weight(1f).fillMaxWidth(),
            fontSize = 20.sp,
            fontWeight = FontWeight.Bold,
            color = Color.Blue
          )
        }
    }
}
```

Código-fonte 5 – Função GameCard
Fonte: Elaborado pelo autor (2023)

Não podemos nos esquecer de ajustar a função “GamesScreen” para que utilize a função “GameCard”. A correção está disponível na listagem “Função GamesScreen utilizando o GameCard”, logo abaixo:

```
. . . trecho de Código omitido
Spacer(modifier = Modifier.height(16.dp))
LazyColumn(){
  items(getAllGames()){
    GameCard(game = it)
  }
}
. . . trecho de Código omitido
```

Código-fonte 6 – Função GamesScreen utilizando a função GameCard
Fonte: Elaborado pelo autor (2023)

Ao executarmos a aplicação, veremos que a IU ficou muito melhor. Note também que a “LazyColumn” já implementa o recurso de rolagem. O resultado deverá se parecer com a figura “Layout da aplicação utilizando a função GameCard”:

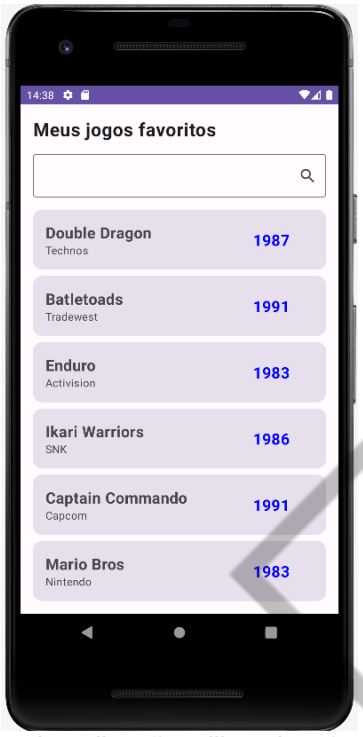


Figura 4 – Layout da aplicação utilizando a função GameCard
Fonte: Elaborado pelo autor (2023)

1.1.2 Implementação da funcionalidade de busca

Para implementarmos a funcionalidade de busca vamos precisar controlar o estado da aplicação, tanto para a digitação no campo de busca quanto para a lista de games. Vamos criar as variáveis de estado no início da função “GamesScreen”, conforme a listagem “Variáveis de estado”:

```
. . . trecho de Código omitido
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun GamesScreen() {

  var searchTextState by remember {
    mutableStateOf("")
  }

  var gamesListState by remember {
    mutableStateOf(getAllGames())
  }
. . . trecho de Código omitido
```

Código-fonte 7 – Variáveis de estado
Fonte: Elaborado pelo autor (2023)

Outro ajuste que deve ser feito é passar o estado para a função “items” da “LazyColumn”. O ajuste pode ser consultado na listagem “Função items utilizando o estado da lista de games”:

```
. . . trecho de Código omitido
Spacer(modifier = Modifier.height(16.dp))
LazyColumn(){
  items(gamesListState){
    GameCard(game = it)
  }
}
. . . trecho de Código omitido
```

Código-fonte 8 – Função items utilizando o estado da lista de games
Fonte: Elaborado pelo autor (2023)

O próximo passo é configurarmos o campo de busca para capturarmos a digitação do usuário, como na listagem “Campo de busca com estado”:

```
. . . trecho de Código omitido
OutlinedTextField(
  value = searchTextState,
  onValueChange = {
    searchTextState = it
  },
  modifier = Modifier.fillMaxWidth(),
  trailingIcon = {
    IconButton(onClick = {}) {
      Icon(
        imageVector = Icons.Default.Search,
        contentDescription = ""
      )
    }
  }
)
. . . trecho de Código omitido
```

Código-fonte 9 – Campo de busca com estado
Fonte: Elaborado pelo autor (2023)

Agora só falta implementarmos a função “onClick” do “IconButton”. A implementação deverá se parecer com a listagem “Função onClick do IconButton”:

```
. . . trecho de Código omitido
OutlinedTextField(
  value = searchTextState,
  onValueChange = {
    searchTextState = it
  },
  modifier = Modifier.fillMaxWidth(),
  trailingIcon = {
    IconButton(onClick = {
      gamesListState = getGamesByStudio(searchTextState)
    }) {
      Icon(
        imageVector = Icons.Default.Search,
        contentDescription = ""
      )
    }
  }
)
. . . trecho de Código omitido
```

Código-fonte 10 – Função onClick do IconButton
Fonte: Elaborado pelo autor (2023)

Execute a aplicação novamente e teste se a busca está funcionando corretamente. O resultado deverá se parecer com a figura “Testando a função de busca”:

Figura 5 – Testando a função de busca
Fonte: Elaborado pelo autor (2023)

Coloque na função “onValueChange” do campo de busca a instrução para buscar o game pelo estúdio. Note que agora a busca ocorre enquanto você digita. Muito mais legal!

1.2 LazyRow

A implementação da “LazyRow” é tão simples quanto a “LazyColumn”. A única diferença é que a nossa lista será apresentada na horizontal.

1.2.1 Implementação da LazyRow

Vamos construir uma lista horizontal com os estúdios da nossa lista de games. Vamos criar uma função chamada “StudioCard” no arquivo “MainActivity.kt”. A implementação desta função está disponível na listagem “Função StudioCard”:

```
@Composable
fun StudioCard(game: Game) {
  Card(modifier = Modifier.size(100.dp).padding(end = 4.dp)) {
    Column(
      verticalArrangement = Arrangement.Center,
      horizontalAlignment = Alignment.CenterHorizontally,
      modifier = Modifier.fillMaxSize()
    ) {
      Text(text = game.studio)
    }
  }
}
```

Código-fonte 11 – Função StudioCard
Fonte: Elaborado pelo autor (2023)

A lista de estúdios ficará logo acima da nossa lista de games. Na função “GamesScreen” faça o ajuste de acordo com a listagem “Incluindo a lista de estúdios”:

```
. . . trecho de código omitido
Spacer(modifier = Modifier.height(16.dp))
LazyRow(){
  items(gamesListState){
    StudioCard(game = it)
  }
}
Spacer(modifier = Modifier.height(16.dp))
LazyColumn() {
  items(gamesListState) {
    GameCard(game = it)
  }
}
. . . trecho de código omitido
```

Código-fonte 12 – Incluindo a lista de estúdios
Fonte: Elaborado pelo autor (2023)

Ao executar a aplicação, teremos uma lista horizontal com os estúdios dos nossos games. A figura “Utilização da LazyRow” nos mostra o resultado final da aplicação:

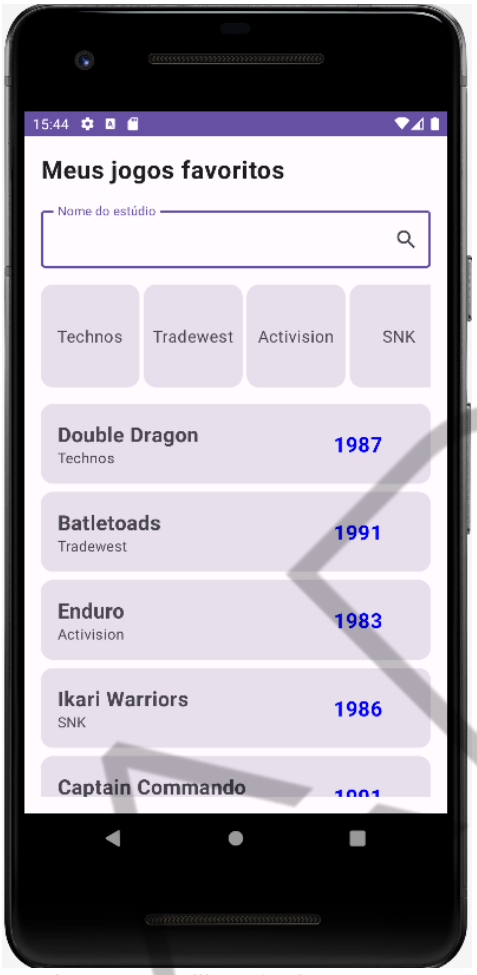


Figura 6 – Utilização da LazyRow
Fonte: Elaborado pelo autor (2023)

2 CONSUMINDO API EXTERNA

2.1 API REST

Uma API REST funciona seguindo o modelo cliente-servidor, ou seja, nós temos um cliente que faz uma requisição solicitando algum recurso e o servidor atende essa requisição fornecendo o recurso requisitado. Os recursos podem ser imagens, arquivos HTML ou simplesmente texto. Geralmente os recursos envolvidos tanto na requisição quanto na resposta em um webservice HTTP usando o padrão REST são textos em formato JSON ou XML. A figura “Estrutura de um webservice REST” nos mostra como é a estrutura básica envolvida na comunicação cliente-servidor.

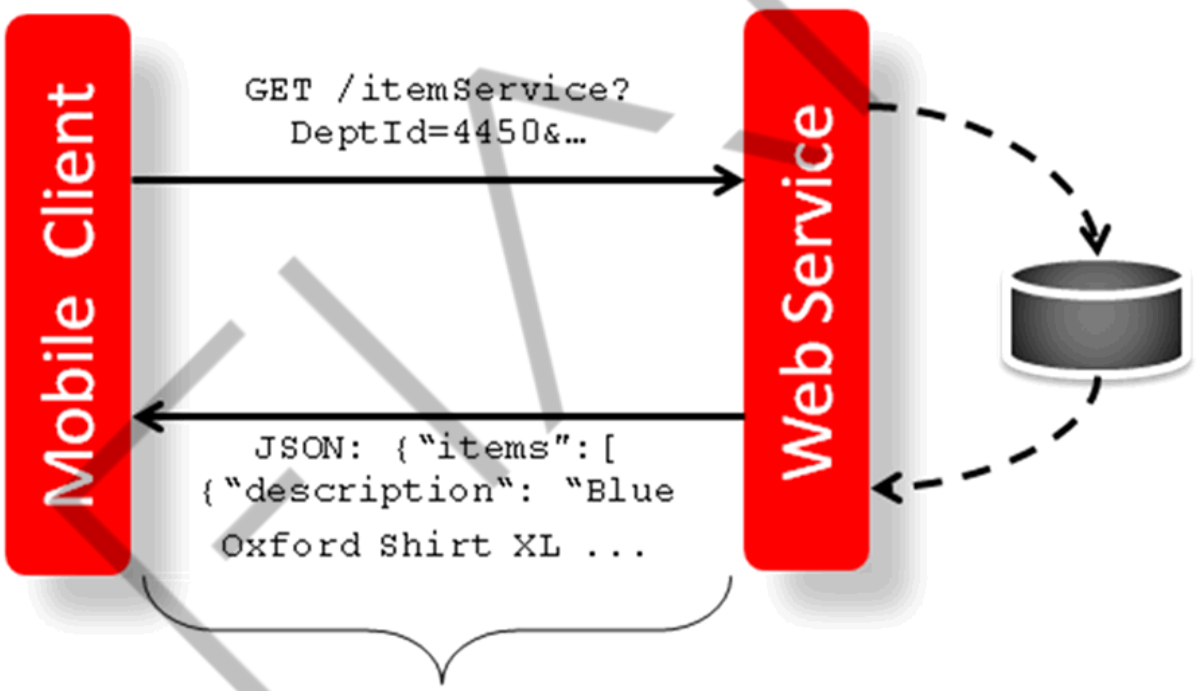


Figura 7 – Estrutura de um webservice REST
Fonte: Oracle docs (2023)

2.2 A biblioteca Retrofit

O Retrofit é uma das bibliotecas mais populares para consumo de APIs REST no universo Android, nos fornecendo uma forma rápida, eficiente e segura de executarmos requisições HTTP e fácil gerenciamento das respostas.

Essa biblioteca nos permite trabalhar com os formatos de dados mais utilizados como o JSON e XML, sendo o JSON o formato dominante. A conversão de objetos para JSON e vice e versa é feito de maneira transparente ao desenvolvedor através da utilização de diversos conversores disponíveis, tais como Gson, Jackson, Moshi etc.

2.3 Projeto Consulta CEP

Nosso projeto realizará a consulta em uma API pública responsável por nos devolver os dados de um endereço a partir de um CEP ou através de uma parte do endereço conhecido, como nome da rua, cidade, estado etc. Utilizaremos a API do ViaCep (https://viacep.com.br/).

Crie um projeto com o nome “Consulta CEP” e apague os métodos “Greeting” e “GreetingPreview”. O código do layout da aplicação está disponível na listagem “Layout do App Consulta CEP”.

```
package br.com.fiap.consultacep

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.heightIn
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Search
import androidx.compose.material3.Card
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
```

```
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.KeyboardCapitalization
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import br.com.fiap.consultacep.ui.theme.ConsultaCEPTheme

class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      ConsultaCEPTheme {
        Surface(
          modifier = Modifier.fillMaxSize(),
          color = MaterialTheme.colorScheme.background
        ) {
          CepScreen()
        }
      }
    }
  }
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun CepScreen() {

  var cepState by remember { mutableStateOf("") }
  var ufState by remember { mutableStateOf("") }
  var cidadeState by remember { mutableStateOf("") }
  var ruaState by remember { mutableStateOf("") }

  Column(
    modifier = Modifier
      .fillMaxWidth()
      .padding(16.dp)
  ) {
    Card(modifier = Modifier.fillMaxWidth()) {
      Column(
        modifier = Modifier
          .fillMaxWidth()
          .padding(16.dp)
      ) {
        Text(text = "CONSULTA CEP", fontSize = 24.sp)
        Text(
          text = "Encontre o seu endereço",
          fontSize = 20.sp
        )
        Spacer(modifier = Modifier.height(32.dp))
        OutlinedTextField(
          value = cepState,
          onValueChange = {
            cepState = it
          },
          modifier = Modifier.fillMaxWidth(),
          label = {
            Text(text = "Qual o CEP procurado?")
          },
          trailingIcon = {
```

```
            IconButton(onClick = { /*TODO*/ }) {
              Icon(
                imageVector = Icons.Default.Search,
                contentDescription = ""
              )
            }
          },
          keyboardOptions = KeyboardOptions(
            keyboardType = KeyboardType.Number
          )
        )
        Spacer(modifier = Modifier.height(32.dp))
        Text(
          text = "Não sabe o CEP?",
          fontWeight = FontWeight.Bold
        )
        Spacer(modifier = Modifier.height(8.dp))
        Row() {
          OutlinedTextField(
            value = ufState,
            onValueChange = {
              ufState = it
            },
            modifier = Modifier
              .weight(1f)
              .padding(end = 4.dp),
            label = {
              Text(text = "UF?")
            },
            keyboardOptions = KeyboardOptions(
              capitalization = KeyboardCapitalization.Characters,
              keyboardType = KeyboardType.Text
            )
          )
          Spacer(modifier = Modifier.height(8.dp))
          OutlinedTextField(
            value = cidadeState,
            onValueChange = {
              cidadeState = it
            },
            modifier = Modifier.weight(2f),
            label = {
              Text(text = "Qual a cidade?")
            },
            keyboardOptions = KeyboardOptions(
              keyboardType = KeyboardType.Text,
              capitalization = KeyboardCapitalization.Words
            )
          )
        }
        Spacer(modifier = Modifier.height(8.dp))
        Row(verticalAlignment = Alignment.CenterVertically) {
          OutlinedTextField(
            value = ruaState,
            onValueChange = {
              ruaState = it
            },
            modifier = Modifier.weight(2f),
            label = {
              Text(text = "O que lembra do nome da rua?")
            },
```

```
            keyboardOptions = KeyboardOptions(
              keyboardType = KeyboardType.Text,
              capitalization = KeyboardCapitalization.Words
            )
          )
          IconButton(onClick = { /*TODO*/ }) {
            Icon(imageVector = Icons.Default.Search, contentDescription =
""")
          }
        }
      }
    }
    Spacer(modifier = Modifier.height(8.dp))
    LazyColumn() {
      items(120) {
        CardEndereco()
      }
    }
  }
}

@Composable
fun CardEndereco() {
  Card(modifier = Modifier
    .fillMaxWidth()
    .padding(bottom = 4.dp)) {
    Column(modifier = Modifier
      .fillMaxWidth()
      .padding(8.dp)
    ) {
      Text(text = "CEP:")
      Text(text = "Rua:")
      Text(text = "Cidade:")
      Text(text = "Bairro:")
      Text(text = "UF:")
    }
  }
}
```

Código-fonte 13 – Layout do App Consulta CEP
Fonte: Elaborado pelo autor (2023)

O layout da nossa aplicação será como o da figura “Layout do App Consulta CEP”:

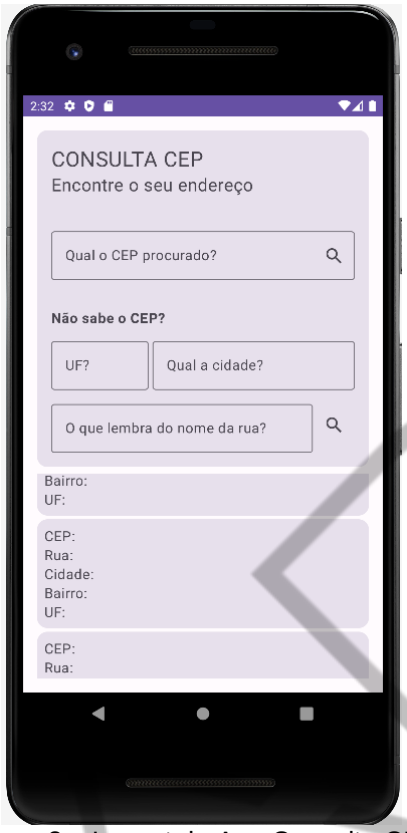


Figura 8 – Layout do App Consulta CEP
Fonte: Elaborado pelo autor (2023)

Crie os pacotes “model” e “service” no pacote principal do projeto. No pacote “model” crie uma classe de dados chamada “Cep”. O código da classe pode ser visualizado na listagem “Classe de dados Endereco”:

```
package br.com.fiap.consultacep.model

data class Endereco(
  val cep: String = "",
  val rua: String = "",
  val cidade: String = "",
  val bairro: String = "",
  val uf: String = ""
)
```

Código-fonte 14 – Classe de dados Endereco
Fonte: Elaborado pelo autor (2023)

2.3.1 Dependências do Retrofit

Vamos acrescentar na sessão “dependencies” do arquivo “build.gradle (Module: app)” as dependências do Retrofit para o nosso projeto. As alterações podem ser consultadas na listagem “Dependências do Retrofit”.

```
dependencies {

    implementation 'androidx.core:core-ktx:1.8.0'
    implementation platform('org.jetbrains.kotlin:kotlin-bom:1.8.0')
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.5.1'
    implementation platform('androidx.compose:compose-bom:2022.10.00')
    implementation 'androidx.compose.ui:ui'
    implementation 'androidx.compose.ui:ui-graphics'
    implementation 'androidx.compose.ui:ui-tooling-preview'
    implementation 'androidx.compose.material3:material3'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-
core:3.5.1'
    androidTestImplementation platform('androidx.compose:compose-
bom:2022.10.00')
    androidTestImplementation 'androidx.compose.ui:ui-test-junit4'
    debugImplementation 'androidx.compose.ui:ui-tooling'
    debugImplementation 'androidx.compose.ui:ui-test-manifest'

    // Dependências do Retrofit
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.9.0'

}
```

Código-fonte 15 – Dependências do Retrofit
Fonte: Elaborado pelo autor (2023)

Não se esqueça de sincronizar as dependências para que possamos utilizar o Retrofit.

A URL para consulta de um CEP no webservice da ViaCep nos retorna o seguinte JSON disponível na listagem “Resposta ViaCep”. Observe que não utilizaremos todos os atributos da resposta. Note também que alguns nomes que utilizamos em nossa classe “Endereco” é diferente do nome do atributo que recebemos do ViaCep. É importante que o nome dos atributos em nossa classe seja igual aos nomes dos atributos do JSON da resposta que recebemos.

```
{
    "cep": "01001-000",
    "logradouro": "Praça da Sé",
    "complemento": "lado ímpar",
    "bairro": "Sé",
    "localidade": "São Paulo",
    "uf": "SP",
    "ibge": "3550308",
    "gia": "1004",
    "ddd": "11",
    "siafi": "7107"
}
```

Código-fonte 16 – Resposta ViaCep
Fonte: Elaborado pelo autor (2023)

Agora que já temos a biblioteca Retrofit disponível em nosso projeto, vamos fazer alguns ajustes em nossa classe de dados “Endereco” para que tudo funcione corretamente. As correções podem ser obtidas na listagem “Classe Endereco corrigida”.

```
package br.com.fiap.consultacep.model

import com.google.gson.annotations.SerializedName

data class Endereco(
  val cep: String = "",
  @SerializedName("logradouro") val rua: String = "",
  @SerializedName("localidade") val cidade: String = "",
  val bairro: String = "",
  val uf: String = ""
)
```

Código-fonte 17 – Classe Endereco corrigida
Fonte: Elaborado pelo autor (2023)

A anotação “@SerializedName” é necessária quando queremos utilizar o nome do atributo da nossa classe diferente do nome do atributo devolvido pela API. Em nosso caso a API devolve o atributo “logradouro”, mas queremos utilizar “rua”. Essa anotação permitirá que o conversor de JSON para objeto consiga localizar os atributos relacionados.

Crie no pacote “service” uma inteface chamada “CepService” com o código disponível na listagem “Interface CepService”:

```
package br.com.fiap.consultacep.service

import br.com.fiap.consultacep.model.Endereco
import retrofit2.Call
import retrofit2.http.GET
import retrofit2.http.Path

interface CepService {

  //https://viacep.com.br/ws/01001000/json/
  @GET("{cep}/json/")
  fun getEndereco(@Path("cep") cep: String): Call<Endereco>

  //https://viacep.com.br/ws/RS/Porto%20Alegre/Domingos/json/
  @GET("{uf}/{cidade}/{rua}/json/")
  fun getEnderecos(
    @Path("uf") uf: String,
    @Path("cidade") cidade: String,
    @Path("rua") rua: String
  ): List<Call<Endereco>>

}
```

Código-fonte 18 – Interface CepService
Fonte: Elaborado pelo autor (2023)

A interface “CepService” possui dois métodos de requisição “GET” por conta da anotação “@GET”. Essa anotação recebe como argumento a parte da URL que é específica para cada requisição. A parte inicial da URL, que chamamos de “Base URL” será configurada posteriormente. Ambos os métodos retornam um objeto do tipo “Call”, que contém a resposta do servidor REST da ViaCep.

A anotação “@Path” indica que a chave que se encontra na URL deverá ser substituída pelo valor do argumento passado na chamada do método.

Crie uma classe chamada “RetrofitFactory” no pacote “service”. Esta classe fará o papel de cliente HTTP, ou seja, ela que fará as requisições para o servidor da ViaCep. O código desta classe está disponível na listagem “Classe RetrofitFactory”:

```
package br.com.fiap.consultacep.service

import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

class RetrofitFactory {

  private val URL = "https://viacep.com.br/ws/"

  private val retrofitFactory = Retrofit
    .Builder()
    .baseUrl(URL)
    .addConverterFactory(GsonConverterFactory.create())
    .build()

  fun getCepService(): CepService {
    return retrofitFactory.create(CepService::class.java)
  }

}
```

Código-fonte 19 – Classe RetrofitFactory
Fonte: Elaborado pelo autor (2023)

A variável “URL” armazena a URL base, ou seja, a parte que é fixa para qualquer requisição ao webservice. O restante da URL será fornecido pela anotação “@GET” da interface que possui os métodos de requisição que serão utilizados.

2.3.2 Executando as chamadas para a API

Vamos executar a chamada para o “endpoint” da ViaCep responsável por nos entregar uma lista de endereços quando fornecemos o estado, a cidade e parte do nome da rua. O método que vamos utilizar será o “getEnderecos()”, que nos devolve uma lista de endereços com base no estado, cidade e rua fornecidos pelo usuário.

Vamos começar criando uma variável de estado que guardará a lista de endereços devolvidos pela API. A inclusão desta variável está disponível na listagem “Variável de estado para a lista de endereços”.

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun CepScreen() {

  var cepState by remember { mutableStateOf("") }
  var ufState by remember { mutableStateOf("") }
  var cidadeState by remember { mutableStateOf("") }
  var ruaState by remember { mutableStateOf("") }

  var listaEnderecoState by remember { mutableStateOf(listOf<Endereco>())
}
```

Código-fonte 20 – Variável de estado para a lista de endereços
Fonte: Elaborado pelo autor (2023)

Agora vamos implementar a chamada “Call” para o servidor da ViaCep. Faremos a chamada no clique do segundo botão “buscar”. O código responsável pela chamada está disponível na listagem “Chamada ao endpoint ViaCep”:

```
Row(verticalAlignment = Alignment.CenterVertically) {
  OutlinedTextField(
    value = ruaState,
    onValueChange = {
      ruaState = it
    },
    modifier = Modifier.weight(2f),
    label = {
      Text(text = "O que lembra do nome da rua?")
    },
    keyboardOptions = KeyboardOptions(
      keyboardType = KeyboardType.Text,
      capitalization = KeyboardCapitalization.Words
    )
  )
  IconButton(onClick = {
    val call = RetrofitFactory().getCepService().getEnderecos(ufState,
cidadeState, ruaState)
    call.enqueue(object : Callback<List<Endereco>>{
      override fun onResponse(
        call: Call<List<Endereco>>,
        response: Response<List<Endereco>>
      ) {
        listaEnderecoState = response.body()!!
      }

      override fun onFailure(call: Call<List<Endereco>>, t: Throwable) {
        TODO("Not yet implemented")
      }

    })
  }) {
    Icon(imageVector = Icons.Default.Search, contentDescription = "")
  }
}
```

Código-fonte 21 – Chamada ao endpoint ViaCep
Fonte: Elaborado pelo autor (2023)

No código em destaque acima criamos um objeto “Call” para o método “getEnderecos” da interface “CepService”, onde passamos o estado, cidade e rua, através das variáveis de estado. Quando efetuamos a chamada, o servidor nos devolverá uma resposta que será armazenada no argumento “response” do método “onResponse” da chamada. Neste momento, atribuímos à variável de estado “listaEnderecoState” o valor retornado pela função “body()” do objeto response, que é a lista de endereço devolvida pelo “endpoint” da ViaCep.

Agora, vamos implementar algumas alterações na função “CardEndereco” para que possamos carregar cada item da “LazyColumn” com os endereços da “listaEnderecoState”. A listagem “CardEndereco final” nos mostra o código final desta função.

```
@Composable
fun CardEndereco(endereco: Endereco) {
  Card(modifier = Modifier
    .fillMaxWidth()
    .padding(bottom = 4.dp)) {
    Column(modifier = Modifier
      .fillMaxWidth()
      .padding(8.dp)
    ) {
      Text(text = "CEP: ${endereco.cep}")
      Text(text = "Rua: ${endereco.rua}")
      Text(text = "Cidade: ${endereco.cidade}")
      Text(text = "Bairro: ${endereco.bairro}")
      Text(text = "UF: ${endereco.uf}")
    }
  }
}
```

Código-fonte 22 – CardEndereco final
Fonte: Elaborado pelo autor (2023)

Para finalizar, vamos corrigir a chamada para a função “CardEndereco” na função “CepScreen” para que a lista seja fornecida de forma correta. A listagem “Passando o estado da lista para CardEndereco” mostra a correção efetuada.

```
Spacer(modifier = Modifier.height(8.dp))
LazyColumn() {
  items(listaEnderecoState) {
    CardEndereco(it)
  }
}
```

Código-fonte 23 – Passando o estado da lista para CardEndereco
Fonte: Elaborado pelo autor (2023)

Antes de executarmos a aplicação, é necessário permitir que o app tenha acesso a rede, então, abra o arquivo “AndroidManifest.xml” e faça a inclusão da permissão de acesso a rede, de acordo com a listagem “Permitir acesso a rede”.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
. . . trecho de código omitido
```

Código-fonte 24 – Permitir acesso a rede
Fonte: Elaborado pelo autor (2023)

Vamos testar a aplicação preenchendo os dados da consulta. Ao clicar no botão de busca, a “LazyColumn” deverá mostrar uma lista com os endereços que atendem aos critérios da busca. A figura “Resultado da consulta” mostra como deverá se parecer a aplicação final.

Figura 9 – Resultado da consulta
Fonte: Elaborado pelo autor (2023)

Nossa aplicação está funcionando e nos devolvendo a lista de endereços. Agora, que tal você implementar a consulta fornecendo o CEP? Fica o desafio para você.

CONCLUSÃO

Toda aplicação deverá, em algum momento, consultar dados em um servidor remoto. Atualmente, as aplicações consomem as mais diversas informações, tais como clima, trânsito, status de voo etc. O Retrofit nos fornece todas as ferramentas necessárias para efetuar o consumo de APIs externar de modo muito prático e fácil.

Também vimos neste capítulo que a construção de listas utilizando “LazyColumn” e “LazyRow” é bastante simples. Explore as funcionalidades que estes composables oferecem.

Agora você já pode construir aplicações que exibem listas organizadas e com dados que podem ser obtidos das mais diversas fontes. Aproveite estes superpoderes!

REFERÊNCIAS

DEVELOPERS. **Listas e Grades**. 2023. Disponível em: <https://developer.android.com/jetpack/compose/lists?hl=pt-br>. Acesso em: 8 jul. 2023.

RETROFIT. **A type-safe HTTP client for Android and Java**. 2023. Disponível em: <https://square.github.io/retrofit/> Acesso em: 8 jul. 2023.