



Integração com API – Consumo de Dados

Objetivo:

Conectar um aplicativo Kotlin com uma API Django REST, consumir os dados dos modelos Client, Product, Employee e Sale, exibir em tela e permitir interação básica com os dados.

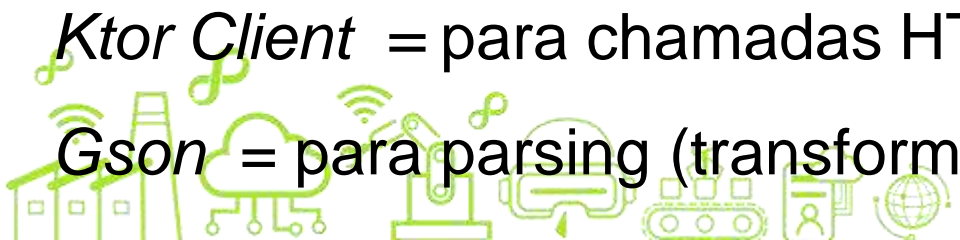
Pré-requisitos e Setup:

API Django pronta e rodando localmente, <http://localhost:8000/clients/>

Dependências necessárias no Kotlin

Ktor Client = para chamadas HTTP

Gson = para parsing (transformar Json em Objetos Kotlin, vice-versa)



Relembrando API

Definição: API = Interface para comunicação entre sistemas.

Funciona com **requisições HTTP** (GET, POST, etc).

Troca dados em formato **JSON**.

Endpoint Django: A API responde com uma lista JSON de objetos.

Cada item representa uma entidade: cliente, produto, etc.

Métodos:

Ler dados com **GET**

Criar dados com **POST**



Relembrando API



tech
lógica

HTTP <http://localhost:8000/clients> Save Share

GET <http://localhost:8000/clients> Send

Params Authorization Headers (7) Body Scripts Tests Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (10) Test Results ↺

200 OK - 73 ms - 914 B - 🌐 ...

{ } JSON ▾ ▶ Preview 🔄 Visualize ▾

```
1  [
2    {
3      "id": 2,
4      "name": "Fulano da Silva",
5      "age": 23,
6      "rg": "123456-7",
7      "cpf": "123456789-01"
8    },
9    {
10     "id": 3,
11     "name": "Siclano da Silva",
12     "age": 41,
13     "rg": "234567-8",
14     "cpf": "234567890-12"
15   },
16 ]
```

CORS: Cross-Origin Resource Sharing

Compartilhamento de Recursos entre Origens Diferentes

É um mecanismo de **segurança** implementado pelos **navegadores** para controlar quais domínios podem acessar recursos de uma **aplicação web** hospedada em outro domínio.

Caso-Exemplo: Se sua Api Django roda em <http://localhost:8000> e seu frontend roda em <http://localhost:3000>. Como são **ORIGENS DIFERENTES**, o **navegador bloqueia automaticamente** qualquer requisição que vá de 3000 para 8000, a menos que a API explicitamente permita isso via CORS.



Instalando CORS no Django

1 – Pip install biblioteca

```
(.venv) PS C:\users\jonatas.copes\Py  
1> pip install django-cors-headers
```

Depois, não esqueça *pip freeze > requirements.txt*

<https://dontpad.com/NDS03/djan>

2 – Adicionar no INSTALLED_APPS e MIDDLEWARE

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'escola.apps.EscolaConfig',  
    'rest_framework',  
    'django_filters',  
    'corsheaders'  
]
```

```
MIDDLEWARE = [  
    'django.middleware.security.SecurityMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
    'corsheaders.middleware.CorsMiddleware',  
    'django.middleware.security.SecurityMiddleware'  
]
```

3 – Alterar ALLOWED_HOSTS,

e adicionar

CORS_ALLOW_ALL_ORIGINS

```
ALLOWED_HOSTS = ['10.0.2.2', 'localhost']
```

```
CORS_ALLOW_ALL_ORIGINS = True
```

Agora NO KOTLIN, Configurar Ktor Client

1. Vá no Graddle > dependencies



```
plugins {  
    alias(libs.plugins.android.application)  
    alias(libs.plugins.kotlin.android)  
    alias(libs.plugins.kotlin.compose)  
    kotlin("plugin.serialization") version "1.9.10"  
}
```

[donthpad.com/NDS03/
kotlin/5/1](https://donthpad.com/NDS03/kotlin/5/1)

```
debugImplementation(libs.androidx.test.manifest)  
implementation("com.google.code.gson:gson:2.10.1")  
implementation("io.ktor:ktor-client-core:3.0.0")  
implementation("io.ktor:ktor-client-android:3.0.0")  
implementation("io.ktor:ktor-client-logging:3.0.0")  
implementation("io.ktor:ktor-client-content-negotiation:3.0.0")  
implementation("io.ktor:ktor-serialization-kotlinx-json:3.0.0")  
implementation("org.slf4j:slf4j-android:1.7.36")  
implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.6.3")  
implementation("org.jetbrains.kotlinx:kotlinx-datetime:0.4.0")  
implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.6.2")  
implementation("androidx.activity:activity-compose:1.7.2")  
implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.7.3")  
implementation("androidx.compose.foundation:foundation:1.5.0")
```

2. Rebuild(Sync)



Com o Ktor Client, podemos começar a fazer GET

1. MODELS > Criar novo modelo

Instant -> lidar com datas

Serializable -> junta obj.

JSON -> obj.Kotlin via
chaves.

OptIn -> indica que usamos
APIs internas/experimentais
de uma biblioteca =

“plugin.serialization”

```
package com.example.test.models

import kotlinx.datetime.Instant
import kotlinx.serialization.Serializable

@OptIn(kotlinx.serialization.InternalSerializationApi::class)
@Serializable
data class Client(
    val id: Int,
    val name: String,
    val age: Int,
    val rg: String,
    val cpf: String
)

@OptIn(kotlinx.serialization.InternalSerializationApi::class)
```


Com o Ktor Client, podemos começar a fazer GET

2. MODELS > Criar novo modelo apiClasses.kt

Instant -> lidar com datas

Serializable -> junta obj.

JSON -> obj.Kotlin via
chaves.

OptIn -> indica uso API
interna/experimental

“plugin.serialization”

Para adicionar as

demaís classes, vá ao

Dontpad.

```
package com.example.test.models

import kotlinx.datetime.Instant
import kotlinx.serialization.Serializable

@OptIn(kotlinx.serialization.InternalSerializationApi::class)
@Serializable
data class Client(
    val id: Int,
    val name: String,
    val age: Int,
    val rg: String,
    val cpf: String
)

@OptIn(kotlinx.serialization.InternalSerializationApi::class)
```

Com o Ktor Client, podemos começar a fazer GET

3. SERVICES > Criar novo serviço ApiService.kt

HttpClient -> é o Ktor instalado

Logging -> ver logs de erros

(se houver)

ContentNegotiation -> como

serializar o JSON:

- ignore chaves desconhecidas
- Mostre bonito o JSON
- permite passar valores

inválidos, não-precisos.

```
object ApiService {  
    val http = HttpClient(Android) {  
        install(Logging) {  
            level = LogLevel.ALL  
        }  
        install(ContentNegotiation) {  
            json(Json {  
                ignoreUnknownKeys = true  
                prettyPrint = true  
                isLenient = true  
            })  
        }  
    }  
}
```



Com o Ktor Client, podemos começar a fazer GET

4. Ainda em SERVICES > apiService.kt

Criar funções de GET: (copiar do Dontpad)

```
suspend fun getClients(): List<Client> {  
    return this.http.get("http://10.0.2.2:8000/clients/").body<List<Client>>()  
}  
  
suspend fun getProducts(): List<Product> {  
    return this.http.get("http://10.0.2.2:8000/products/").body<List<Product>>()  
}  
  
suspend fun getEmployees(): List<Employee> {  
    return this.http.get("http://10.0.2.2:8000/employees/").body<List<Employee>>()  
}  
  
suspend fun getSales(): List<SaleGet> {  
    return this.http.get("http://10.0.2.2:8000/sales/").body<List<SaleGet>>()  
}
```



Com o Ktor Client, podemos começar a fazer GET

5. VIEWS > Criar nova view apiClient.kt

5.1. Primeiro, Cria-se o item

Da Lista

Tipo Cartão,

Contendo o nome, idade

(em coluna)

(depois colocaremos

mais informações)

```
@Composable
fun ClientListItem(client: Client) {
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .padding(horizontal = 16.dp, vertical = 4.dp),
        elevation = CardDefaults.cardElevation(4.dp)
    ) {
        Column(
            modifier = Modifier.padding(16.dp),
        ) {
            Text(
                text = client.name,
                style = MaterialTheme.typography.bodyLarge
            )
            Text("${client.age}")
        }
    }
}
```



Com o Ktor Client, podemos começar a fazer GET

5. Ainda em VIEWS > apiClientView.kt

5.2. Cria-se a lista dos itens

do passo anterior

Lista de Cartões,
LazyColumn

Veja no dontpad.

```
@Composable
fun ClientList(clients: List<Client>){
    LazyColumn (
        modifier = Modifier.fillMaxSize(),
        contentPadding = PaddingValues(vertical = 8.dp)
    ) {
        items(clients) { client ->
            ClientListItem(client)
        }
    }
}
```



Com o Ktor Client, podemos começar

6. VIEWS > Crie nova view SwipePager.kt

Essa view será o paginador do nosso App.

Recebe os Clientes, Produtos, Employees, Sales.

Atribui um numero de página pra cada um.

Acessa essa página pelo número.

Redireciona pra View que fizemos no passo anterior

(ATUAL-TESTE) Todas vão pro mesmo lugar:

ClientList(clientes), depois mudaremos pras outras views que vamos criar.

```
@Composable
fun SwipePagerScreen(
    clients: List<Client>,
    products: List<Product>,
    employees: List<Employee>,
    sales: List<SaleGet>
) {
    val pageCount = 4
    val pagerState = rememberPagerState(
        initialState = 0,
        pageCount = { pageCount }
    )
    HorizontalPager(
        state = pagerState,
        modifier = Modifier.fillMaxSize()
    ) { page ->
        when (page) {
            0 -> ClientList(clients)
            1 -> ClientList(clients)
            2 -> ClientList(clients)
            3 -> ClientList(clients)
        }
    }
}
```



Com o Ktor Client, podemos começar a fazer GET

7. Atualizar a Main

Usa **Remember**,
estado mutável
começando vazio

LaunchedEffect
faz a chamada
GET

Usa o **Swiper** do
passo anterior.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    var clients by remember { mutableStateOf<List<Client>>(emptyList()) }
    var products by remember { mutableStateOf<List<Product>>(emptyList()) }
    var employees by remember { mutableStateOf<List<Employee>>(emptyList()) }
    var sales by remember { mutableStateOf<List<SaleGet>>(emptyList()) }

    LaunchedEffect(Unit) {
        clients = ApiService.getClients()
        products = ApiService.getProducts()
        employees = ApiService.getEmployees()
        sales = ApiService.getSales()
    }

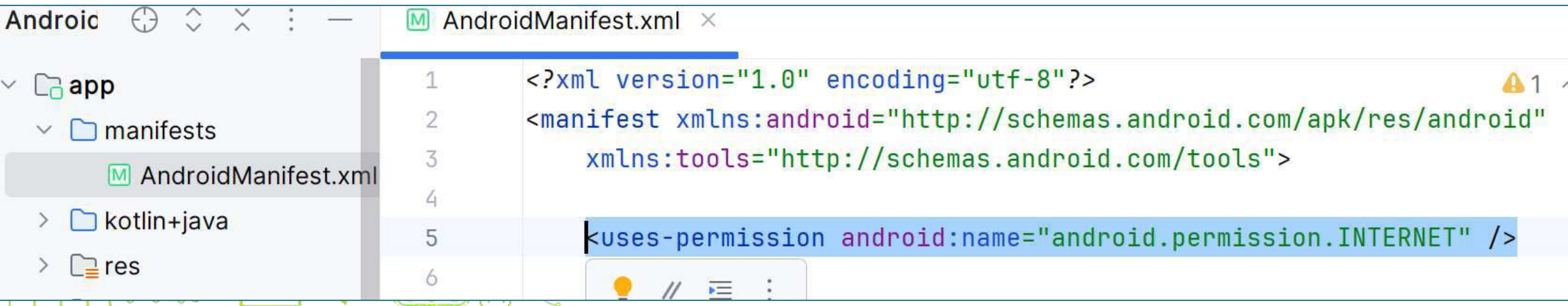
    SwipePagerScreen(clients, products, employees, sales)
}
```

Habilitar no Android Manifest

8. Android Manifest

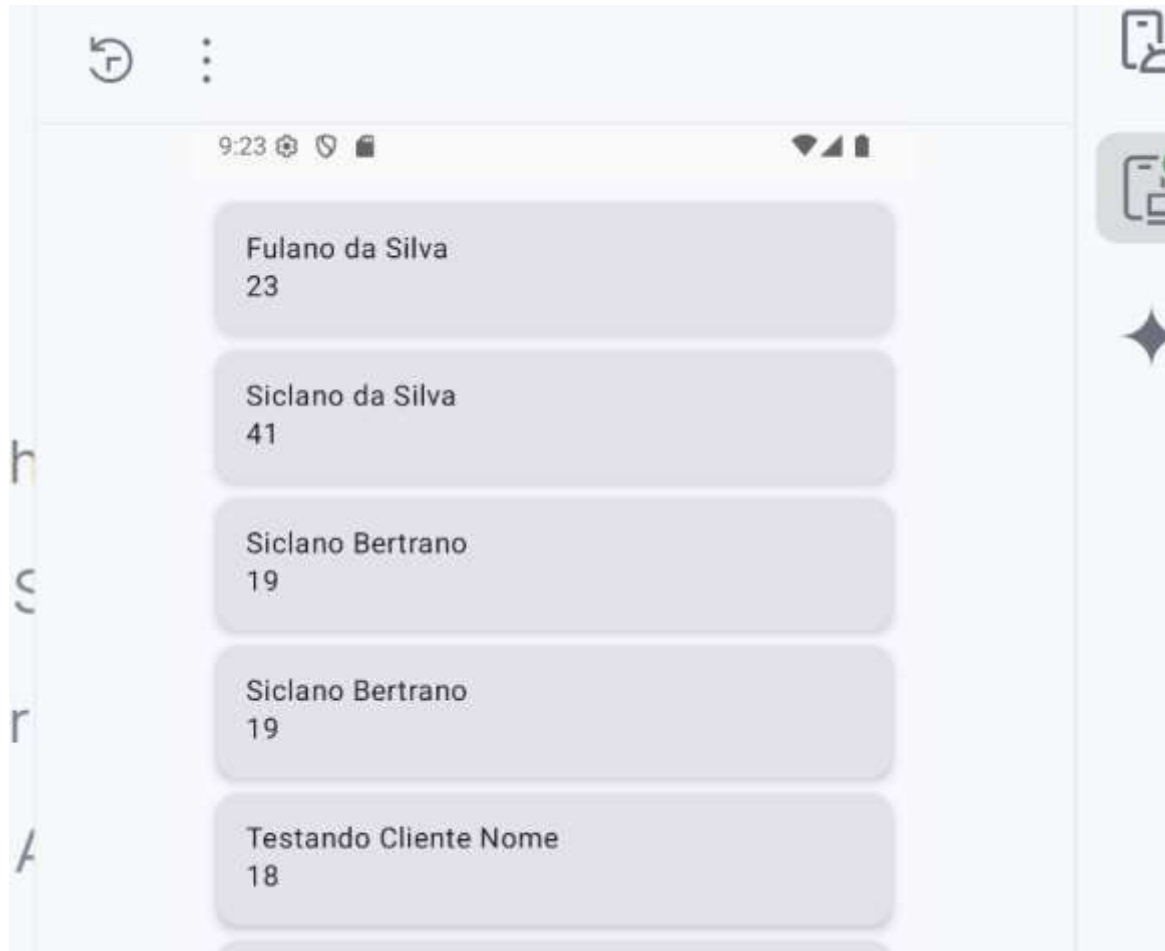
Qualquer feature que seu app utilizar no Android deve ser registrada no manifest. Se você for usar a Câmera, GPS, internet ou qualquer outro recurso do dispositivo. Isso é política do app para ter upload na Play Store.

Abaixo, adicionamos permissão pra usar internet.



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools">
4
5     <uses-permission android:name="android.permission.INTERNET" />
6
```


Funciona



Hora de fazer os outros

GETS:

1. Melhorar:

- Clientes(mais info)

2. Criar Views:

- apiEmployeeView

- apiProductView

- apiSaleView

3. Atualizar Swiper

Colocar números

4. Rodar



@fpftech.educacional

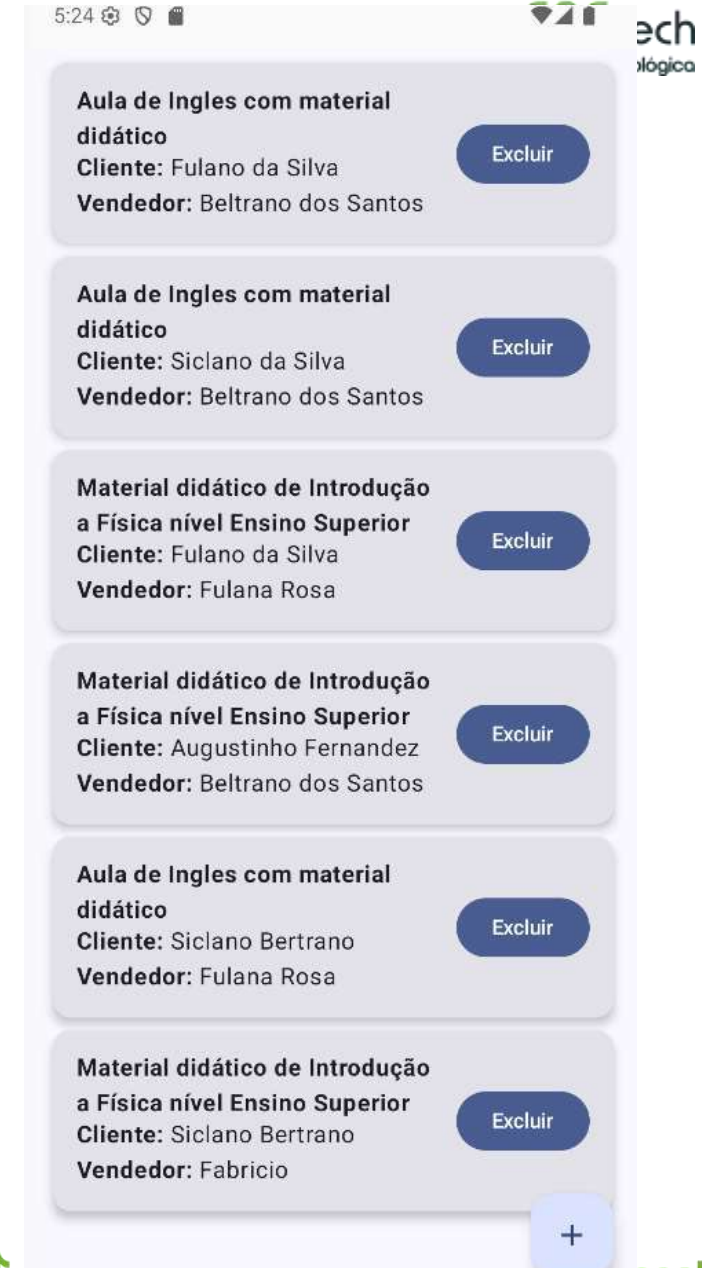
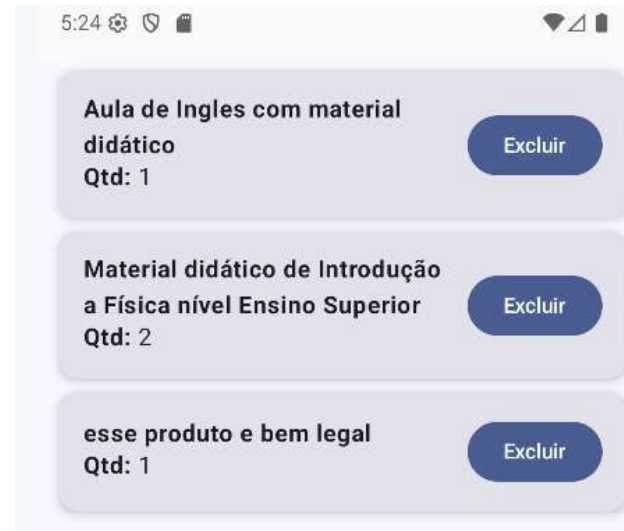


<https://dontpad.com/NDS03/kotlin/5/get/1>

Nos só
arrumamos o
GET pra ficar
mais bonito

POST ainda não
funciona

DELETE também
ainda não funciona



Vamos fazer CLIENTS funcionar o post e o delete.

1 – Na APIservice criamos endpoints de POST e DELETE para o cliente.

2 - No client view (criamos um Formulário com validadores)

3 – No swiper Adicionamos o delete.

4 – Na main adicionamos a chamada do formulário e juntamos o



41 anos
CPF: 234567890-12
RG: 234567-8

Siclano Bertrano
19 anos
CPF: 234456859-48
RG: 12345498-9

Siclano Bertrano
19 anos
CPF: 234456859-48
RG: 12345498-9

Bernardo
19 anos
CPF: 12334123-34
RG: 1231029-3

Augustinho Fernandez
38 anos
CPF: 568590987-65
RG: 456789112-1

Teste Kotlin

Deve ser entre 18 e 100

Caracteres restantes: 12

Caracteres restantes: 12

CPF: 123456789-01
RG: 123456-7

Siclano da Silva
41 anos
CPF: 234567890-12
RG: 234567-8

Siclano Bertrano
19 anos
CPF: 234456859-48
RG: 12345498-9

Siclano Bertrano
19 anos
CPF: 234456859-48
RG: 12345498-9

Bernardo
19 anos
CPF: 12334123-34
RG: 1231029-3

Augustinho Fernandez
38 anos
CPF: 568590987-65
RG: 456789112-1

Teste Kotlin
26 anos

+



Obrigado!



    @fpftech.educacional