

Memorando

Aplicações Móveis, 2025/2026

De: Holog António

Nº de Matrícula: 20211211

Assunto: LAB. 1 - Introdução ao Android

Data: 14 de Outubro de 2025

1. Introdução

A experiência laboratorial teve como foco a iniciação ao desenvolvimento de aplicações móveis para a plataforma Android. O foco principal não foi apenas na implementação da funcionalidade, mas no mapeamento e compreensão da arquitetura de build e de programação.

O contexto original do Laboratório #01 focou na iniciação ao desenvolvimento Android utilizando o ambiente **Android Studio**, a linguagem **Java** e o sistema de build **Gradle**. Os objetivos centrais incluíram a familiarização com componentes nucleares como Activity e Intent para navegação, a implementação da interface gráfica (UI) através de ficheiros XML e a gestão de listas dinâmicas com os padrões imperativos de ArrayList, ListView e ArrayAdapter. Esta abordagem sublinha uma arquitetura de plataforma-específica, onde a lógica de negócio (Java) está acoplada à estrutura de Views nativas e o fluxo de dados exige manipulação manual de adaptadores para refletir as alterações na UI.

A experiência adaptada mudou drasticamente o foco para o ambiente cross-platform **React Native e Expo**, utilizando **JavaScript/TypeScript** e **Node.js/npm/Metro Bundler**. O objetivo não foi apenas replicar a funcionalidade, mas mapear os conceitos: as Activities tornaram-se Componentes Funcionais React, o XML foi substituído por JSX e a navegação por Intents foi substituída pelo navigation.navigate() do React Router. Esta adaptação visou demonstrar a flexibilidade do desenvolvimento moderno e a capacidade de abstrair as complexidades nativas do Android através de um *framework* declarativo.

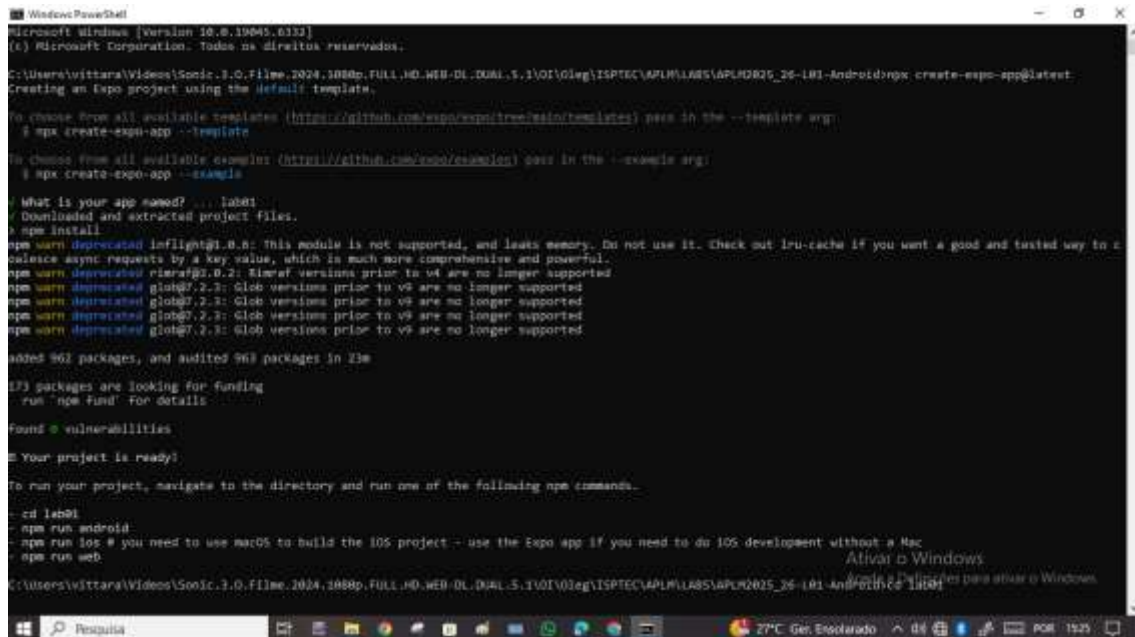
No que tange às ferramentas de *build* e gestão de projeto, o complexo sistema **Gradle** com a sua estrutura de múltiplos ficheiros build.gradle e o settings.gradle foi traduzido para o sistema Node.js/npm. O ficheiro package.json assumiu o papel de gestor de dependências e de *build*, definindo *scripts* de execução e implementando a lógica de Múltiplos Projetos através de **npm Workspaces** (Monorepo). Esta substituição simplificou a configuração do *build* para a maioria dos casos de uso, embora a experiência tenha revelado que o npm start se tornou sensível a problemas de rede/Firewall, gerando erros como o TypeError: fetch failed.

A observação mais significativa da experiência reside no contraste entre o paradigma **Imperativo** (Java/Android) e **Declarativo** (React Native). Enquanto o Java exige que o programador instrua o sistema passo a passo (ex: usar findViewById e chamar notifyDataSetChanged explicitamente), o React Native utiliza o conceito de Estado (useState), onde as alterações nos dados (ArrayList) desencadeiam automaticamente a re-renderização da interface do utilizador, simplificando significativamente a gestão do ciclo de vida dos componentes e a reatividade da aplicação (como demonstrado no Exercício da Lista de Tarefas).

2. Experiências Realizadas

I (Hello World) – Componentes e Navegação Reativa

A `Activity` é substituída por **Componentes Funcionais React**. O `Intent` é mapeado para `router.push()` (React Router), passando dados via `route.params`. A UI é definida por **JSX** e estilizada por `StyleSheet`. A vinculação de dados é resolvida pelo **Estado** (`useState`), eliminando `findViewById`.



```
Microsoft Windows [version 10.0.19045.6333]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\vitara\Videos\Gonç.3.O.Filme.2024.3888p.FULL.HD.WEB-DL.DUAL.S.1\OI\Oleg\ISPTEC\APLWLABS\APLMD825_26-181-Android>npx create-expo-app@latest
Creating an Expo project using the default template.

To choose from all available templates (https://github.com/expo/expo/tree/main/templates) pass in the --template arg:
  npx create-expo-app --template

To choose from all available examples (https://github.com/expo/examples) pass in the --example arg:
  npx create-expo-app --example

What is your app named? ... lab01
  Downloaded and extracted project files.
  npm install
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and tested way to c
npm warn deprecated rimraf@3.0.2: rimraf versions prior to v4 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
added 961 packages, and audited 963 packages in 23s

137 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

Your project is ready!

To run your project, navigate to the directory and run one of the following npm commands:

  cd lab01
  npm run android
  npm run ios # you need to use macOS to build the iOS project - use the Expo app if you need to do iOS development without a Mac
  npm run web

C:\Users\vitara\Videos\Gonç.3.O.Filme.2024.3888p.FULL.HD.WEB-DL.DUAL.S.1\OI\Oleg\ISPTEC\APLWLABS\APLMD825_26-181-Android>
```

Figura 1 - Criar um projecto React Native

Mapeamento de Ficheiros (Android/Java para React Native/Expo)

No desenvolvimento React Native/Expo, a estrutura de diretórios é simplificada e a configuração nativa (Android) é abstraída para ficheiros JavaScript/JSON de alto nível.

| Ficheiro Original (Android/Java) | Equivalente no React Native/Expo |
|---|--|
| app/src/main/res/layout/activity_main.xml | src/screens/HomeScreen.tsx (ou qualquer componente) |

Definição da Interface do Utilizador (UI). Em vez de XML, o *layout* é definido usando **JSX** (mistura de JavaScript e marcação) dentro de Componentes React. A estilização (substituindo o XML de atributos) é feita via `stylesheet` (em JavaScript) e **Flexbox** para o posicionamento.

app/src/main/java/.../MainActivity.java

src/screens/HomeScreen.tsx
(ou outro componente funcional)

Lógica e Controlo. Substituído por um **Componente Funcional React** ou uma classe JavaScript/TypeScript. Contém os *hooks* (*useState*, *useEffect*) para gerir o **Estado** e a **Lógica de Negócio** que reage aos eventos da UI (ex: *onPress*).

app/src/res/AndroidManifest.xml

app.json (ou *app.config.js*)

Configuração da Aplicação. O Expo abstrai este ficheiro nativo. As permissões, o nome da aplicação, o ícone e a orientação do ecrã são definidos no ficheiro *app.json* (JSON). Durante o *build* (ex: para APK), o Expo injeta esta configuração no *AndroidManifest.xml* nativo final.

app/build.gradle

package.json

Gestão de *Build* e Dependências. Este ficheiro JavaScript/JSON define as dependências externas e internas do projeto (na secção "dependencies"). Também contém os *scripts* de execução ("*start*", "*android*") que são usados para iniciar o **Metro Bundler** e orquestrar o processo de *build*.

Resumo da Árvore de Diretórios (Visualização Adaptada)

No terminal ou IDE, a estrutura de um projeto React Native/Expo que contém os elementos acima teria esta aparência:

```
my-app/
├── package.json      <-- Substitui app/build.gradle
├── app.json          <-- Substitui AndroidManifest.xml
├── node_modules/     <-- Dependências (substitui o cache do Gradle)
├── src/
│   └── screens/
│       └── HomeScreen.tsx <-- Substitui MainActivity.java e o XML de
layout
```

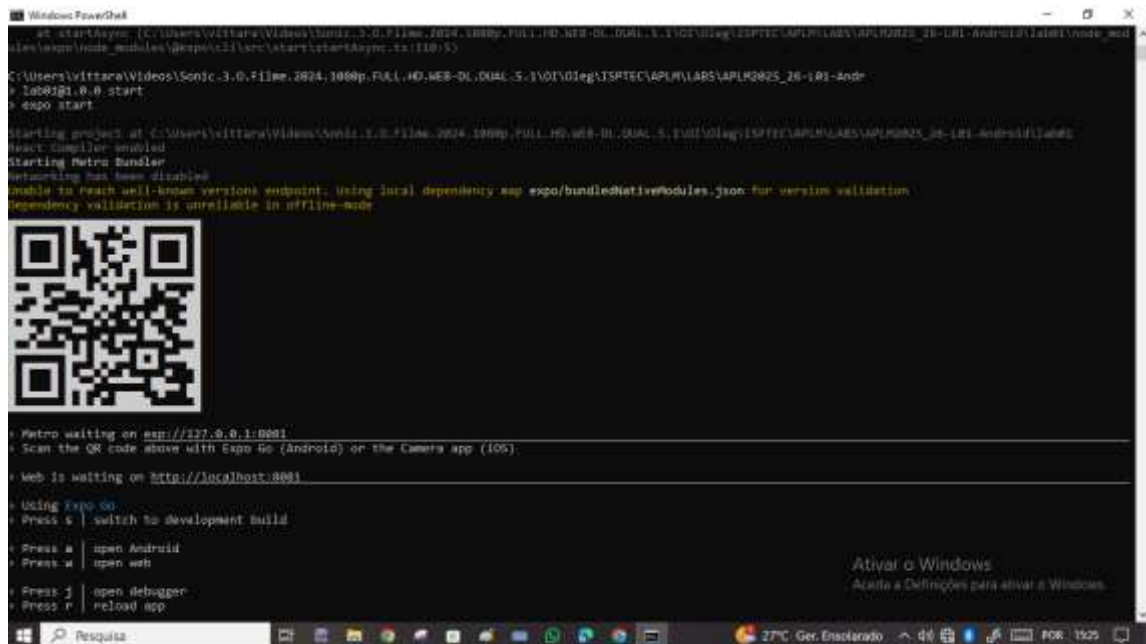


Figura 2 - Executar a aplicação

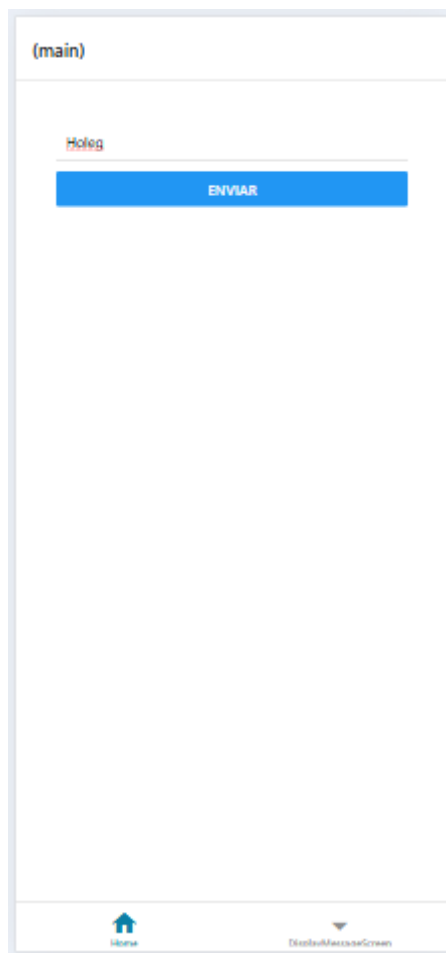


Figura 3 - Exercício 1 (Tela 1).

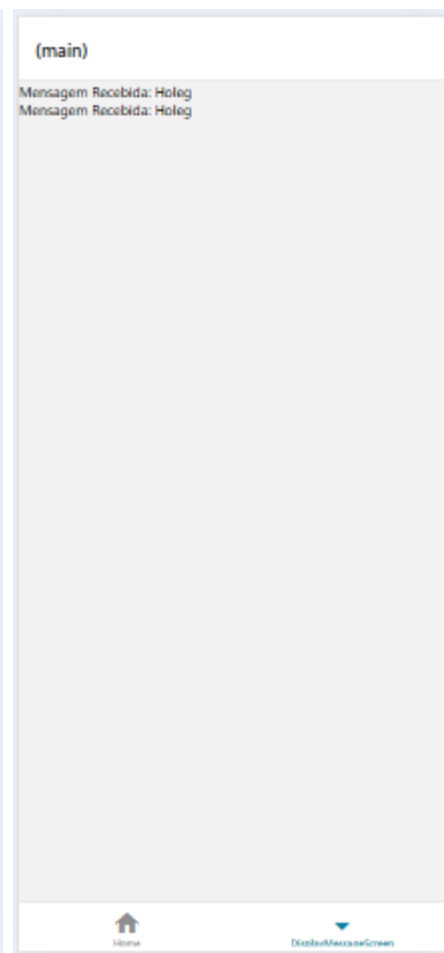


Figura 4 - Exercício 1 (Tela 2)

Resposta às Questões (Adaptadas a React Native)

As questões originais estão enraizadas no desenvolvimento Android/Java. As respostas abaixo fornecem o conceito equivalente e como a tarefa é realizada no ecossistema React Native.

1. O quê que alterou no ficheiro `AndroidManifest.xml`?

- **Resposta Adaptada (React Native/Expo):** Geralmente, **nada**. O desenvolvimento em React Native, especialmente com Expo, abstrai o programador da necessidade de modificar diretamente os ficheiros de projeto nativos (`AndroidManifest.xml` para Android e `Info.plist` para iOS).
- **Configuração Equivalente:** A maior parte da configuração da aplicação (nome, ícone, permissões básicas) é feita no ficheiro **`app.json`** ou **`app.config.js`** do React Native. Se a aplicação for *ejectada* para o React Native CLI ou se forem necessários *módulos nativos* personalizados, o `AndroidManifest.xml` pode ser alterado para adicionar novas permissões ou declarar novas *Activities* nativas (o que é raro no fluxo de trabalho de *Activities*).

2. Como o evento clique é tratado?

- **Resposta Adaptada (React Native/JSX):** O evento de clique é tratado através da *propriedade* **`onPress`** do componente.
- **Mecanismo:** Em vez de declarar um método `android:onClick="sendMessage"` no XML e defini-lo em Java, o `onPress` do componente `<Button>` (ou qualquer componente **`Touchable`**) aceita uma **função JavaScript** que é executada quando o utilizador toca no elemento. Esta é a forma padrão do React de tratar eventos.

3. Como identificar e obter ou definir componentes de interface do código Java?

- **Resposta Original (Android):** Em Java, utilizava-se o método `findViewById(R.id.edit_message)` (onde `R.id.edit_message` era a referência criada a partir do XML) para obter o objeto `View` e, em seguida, manipulá-lo.
- **Resposta Adaptada (React Native/JavaScript):** Em React Native, evita-se a manipulação direta do DOM/View nativa. O código JavaScript interage com a UI principalmente através de dois mecanismos:
 - **State (Estado):** Para obter e definir valores em componentes de formulário (como `TextInput`), utiliza-se o **Estado do Componente** (`useState`). Quando o state muda, o React **re-renderiza** o componente, atualizando a UI. (Ex: o `value={text}` do `TextInput` é o que define o seu conteúdo).
 - **Refs (Referências):** Para tarefas específicas que requerem acesso direto ao elemento (ex: focar um campo), utiliza-se o

sistema **useRef** do React para obter uma referência programática ao componente.

4. Qual é a finalidade da intent da variável criada na classe **DisplayMessageActivity**?

- **Resposta Original (Android):** A Intent é um objeto de mensagem que permite a comunicação entre componentes (ex: uma Activity a iniciar outra). É utilizada para anexar dados (**Extras**) que a Activity de destino (**DisplayMessageActivity**) pode ler.
- **Resposta Adaptada (React Native/React Navigation):** O conceito de Intent é substituído pelas funções de navegação da biblioteca **React Navigation** (ou equivalente – expo-router).
 - **Finalidade:** O objeto **navigation** (que substitui a Intent de chamada) permite iniciar a transição para uma nova tela (`navigation.navigate('NomeDoEcrã')`) e, o mais importante, permite passar um objeto de dados (os **params** ou parâmetros) para a tela de destino.
 - **Leitura dos Dados:** A tela de destino acede a estes dados através do objeto **route** (o objeto `route.params` é o equivalente a ler os Extras da Intent).

II (Gradle) – Gestão de *Build* Node.js/npm

O `package.json` substitui o `build.gradle`, definindo dependências e *scripts* de execução (`npm start`). O **Metro Bundler** é o motor de *build* que transpila (via Babel) e *bundle* o código. A modularidade de Múltiplos Projetos (Gradle) é alcançada com **npm Workspaces**, que liga pacotes locais (módulos) através do campo `workspaces` no `package.json` raiz.

3.1. Node.js e Gerenciadores de Pacotes (npm/yarn)

O **Node.js** e o **npm** (Node Package Manager) ou **yarn** são a espinha dorsal do desenvolvimento em JavaScript moderno e, consequentemente, em React Native. Eles substituem o repositório Maven/JCenter e a gestão de dependências do Gradle.

- **Comandos básicos (em substituição a `gradle -v` ou `gradle -q tasks`):**
 - **Verificar a versão:** `node -v` e `npm -v` (ou `yarn -v`).
 - **Instalar uma dependência:** `npm install [nome-do-pacote]` (ou `yarn add [nome-do-pacote]`). Por exemplo: `npm install react-navigation/stack`
 - **Verificar dependências:** Inspeção o ficheiro `package.json` na raiz do projeto. Este ficheiro lista todas as dependências do projeto, divididas em `dependencies` (para o código de produção) e `devDependencies` (para ferramentas de desenvolvimento). Este é o equivalente ao `build.gradle` a nível de gestão de bibliotecas.

- **Tarefas de Script:** O package.json também contém uma secção scripts que define comandos de *build* e execução, como start, android, ios. Estes comandos substituem as tarefas (tasks) básicas do Gradle.

3.2. Simples projecto Java com Gradle Java plugin (Adaptado para React Native/Node.js)

```

C:\Windows\System32\cmd.exe
Microsoft Windows [Versão 10.0.19045.0132]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2

C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android>cd exemplo2

C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2>npm init -y
npm notice to C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2\package.json:

{
  "name": "exemplo2",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}

C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2>mkdir src
C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2>touch src\Quote.js
"touch" is not recognized as an internal or external command,
operable program or batch file.

C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2>type nul > src\Quote.js
C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2>cd ..
C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android>cd exemplo2
C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2>type nul > src\Quote.js
C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2>dir
C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2>dir

```

Figura 5 - Criar directorios e ficheiros para projecto exemplo2 em react native

```

C:\Windows\System32\cmd.exe
Microsoft Windows [Versão 10.0.19045.0132]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2>src
C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2>touch src\Quote.js
"touch" is not recognized as an internal or external command,
operable program or batch file.

C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2>type nul > src\Quote.js
C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2>cd ..
C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android>cd exemplo2
C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2>type nul > src\Quote.js
C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2>dir
Volume in drive C has no label.
Volume Serial Number is 000C-703B

Directory of C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2

14/10/2025  03:06    <DIR>          .
14/10/2025  03:06    <DIR>          ..
14/10/2025  02:53                222 package.json
14/10/2025  03:06                0 run.js
14/10/2025  02:55    <DIR>          src
                2 File(s)    222 bytes
                3 Dir(s)  10.607.170.340 bytes free

C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2>run
exemplo2@1.0.0 run:logic
> node run.js

Aplicação Quote em execução (Script Mode.js)
Quote { id: 1, author: 'João Costa', text: 'Deus é vida.' }
Resultado do toString(): [Citação ID: 1] - "Deus é vida." por João Costa
Autor alterado: Fernando Pessoa

C:\Users\vittara\Videos\Sonic.3.0.Filme.2024.1080p.FULL.HD.MEB-DL.DUAL.5.1\OI\OIeg\ISPTEC\APLM\LABS\APLM2025_26-101-Android\exemplo2>

```

Figura 6 - execução do projecto exemplo2

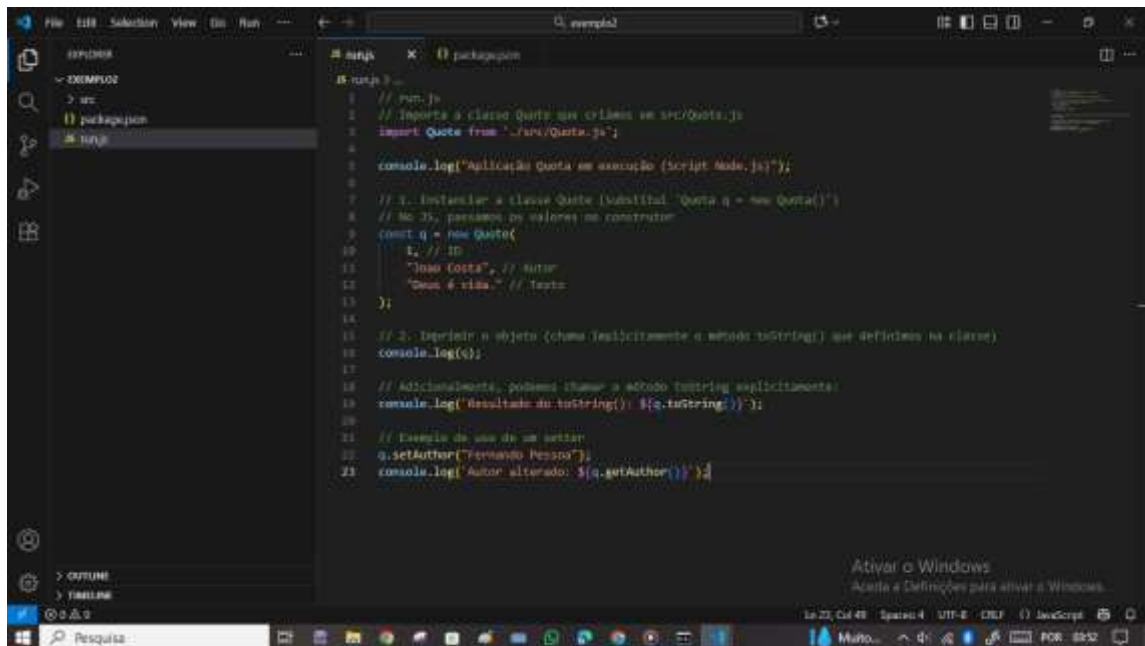


Figura 7 - ficheiro de execução de lógica de projecto exemplo2

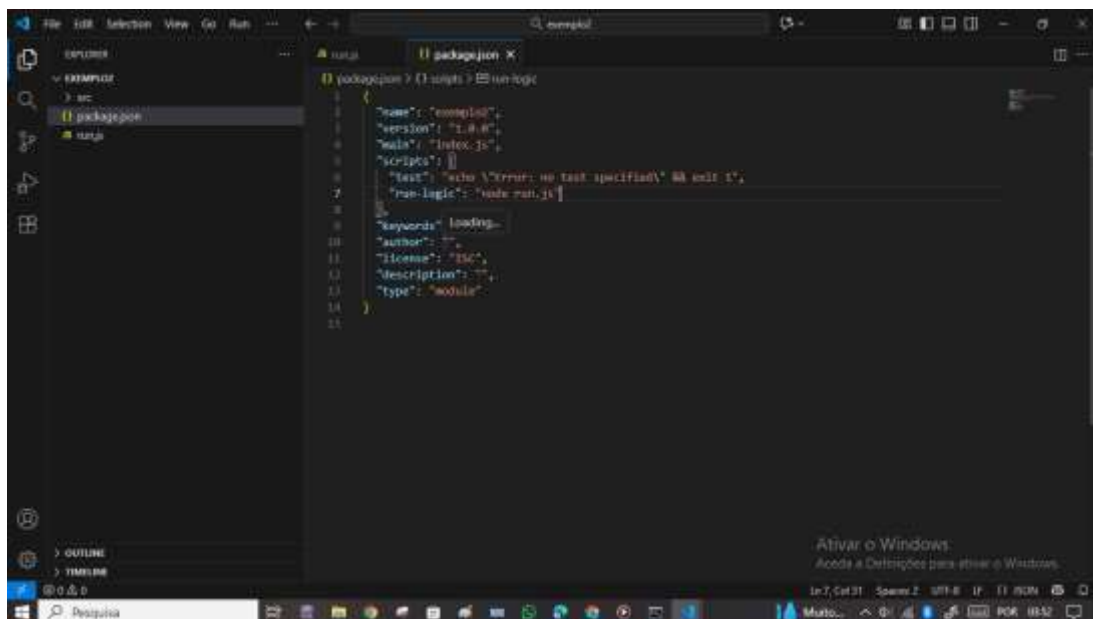


Figura 8 - Definição de script de execução do projecto exemplo2

3.3. Múltiplos Projetos Java (Análise do Exemplo Gradle)

Estrutura do Projeto Gradle

O projeto está dividido em quatro níveis de configuração/execução:

1. **Ficheiro Raiz:** settings.gradle e build.gradle (na raiz).
2. **Subprojetos/Diretórios:** app, api, e common.

Análise e Resposta às Questões

1. Em quais subprojetos este projeto está dividido?

O projeto está dividido nos seguintes subprojetos (determinados pela declaração `include` no `settings.gradle`):

Subprojeto Função Provável

| | |
|----------------------|--|
| <code>:app</code> | O projeto principal/executável (a aplicação final que usa os outros módulos). |
| <code>:api</code> | O projeto que define a interface de programação (API) ou contrato de serviço, usando a lógica <code>common</code> . |
| <code>:common</code> | O projeto que contém a lógica de domínio ou utilidades que são reutilizáveis por outros módulos (<code>app</code> e <code>api</code>). |

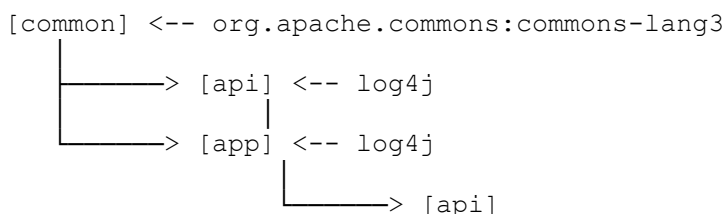
Como executar o Gradle para construir:

| Objetivo | Comando Gradle | Explicação |
|--------------------------------------|---|---|
| (i) Todo o projeto | <code>gradle build</code> | Executado na raiz do projeto. O Gradle Wrapper (<code>gradlew</code> ou <code>gradle</code>) irá processar o <code>build.gradle</code> raiz, que aplica a tarefa <code>build</code> recursivamente a todos os subprojects (<code>app</code> , <code>api</code> , <code>common</code>) definidos. |
| (ii) Cada subprojeto individualmente | <code>gradle :[nome_do_subprojeto]:build</code> | Deve ser executado na raiz. Por exemplo: <code>gradle :app:build</code> , <code>gradle :api:build</code> , OU <code>gradle :common:build</code> . Isto foca a execução apenas naquele subprojeto e nas suas dependências. |

As **dependências** são extraídas dos ficheiros `build.gradle` de cada subprojeto:

| Subprojeto | Dependências de Terceiros (Externas) | Dependências Locais (Projeto) |
|----------------------|--|--|
| <code>:common</code> | <code>org.apache.commons:commons-lang3:3.3.2</code> | Nenhuma |
| <code>:api</code> | <code>org.apache.commons:commons-lang3:3.3.2</code> , <code>log4j:log4j:1.2.17</code> | <code>:common</code> |
| <code>:app</code> | <code>log4j:log4j:1.2.17</code> | <code>:common</code> , <code>:api</code> |

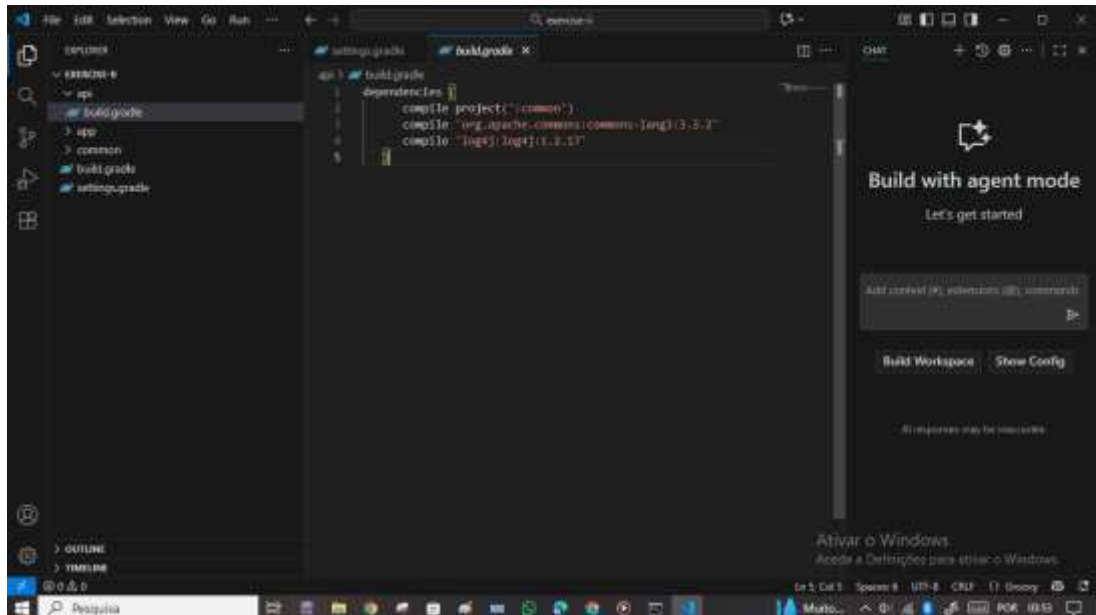
Gráfico de Dependências:



Resumindo:

- `common` é o módulo base, dependente apenas de uma biblioteca externa.
- `api` depende da lógica de `common` e de duas bibliotecas externas.

- `app` depende do módulo de lógica `common` e do módulo de interface `api`, além da biblioteca `log4j`.



III (Lista de Tarefas) – Gestão de Dados Declarativa

O `ListView` com `ArrayAdapter` é substituído pelo componente `<FlatList>`. O modelo de dados (`ArrayList`) é gerido pelo `useState<Todo[]>`. A inserção de novos itens é **reativa**: a função `setTodos` é chamada com um novo *array* (mutação não destrutiva). O React, ao detetar a mudança de **Estado**, força a re-renderização automática da `<FlatList>`, eliminando a necessidade de chamadas manuais (`notifyDataSetChanged()`).

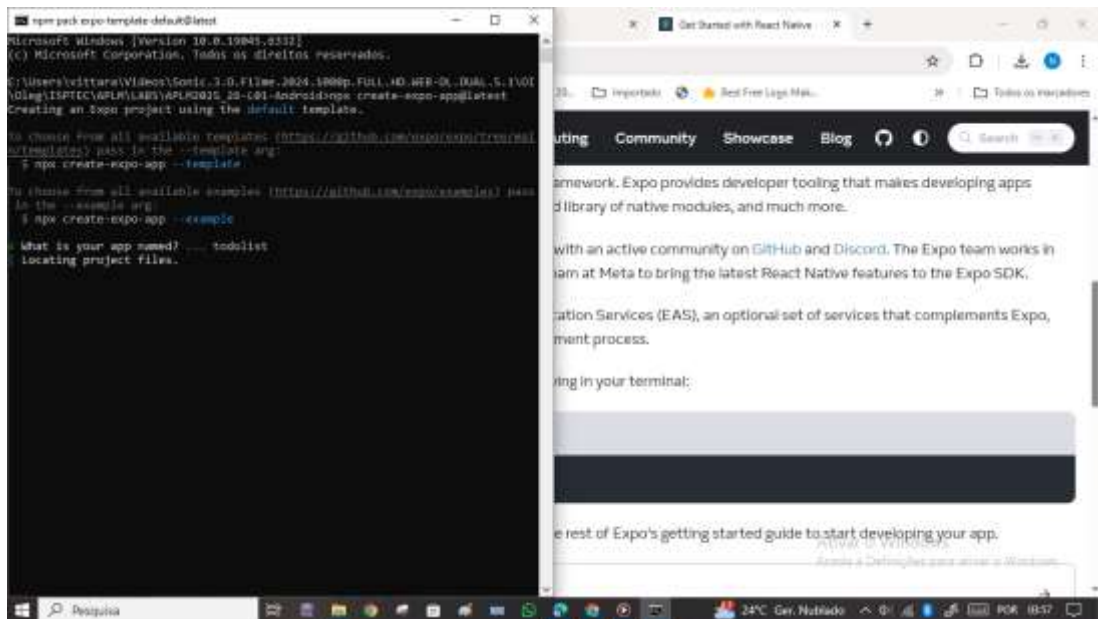


Figura 9 - Criação do projecto todoList (Exercício III).

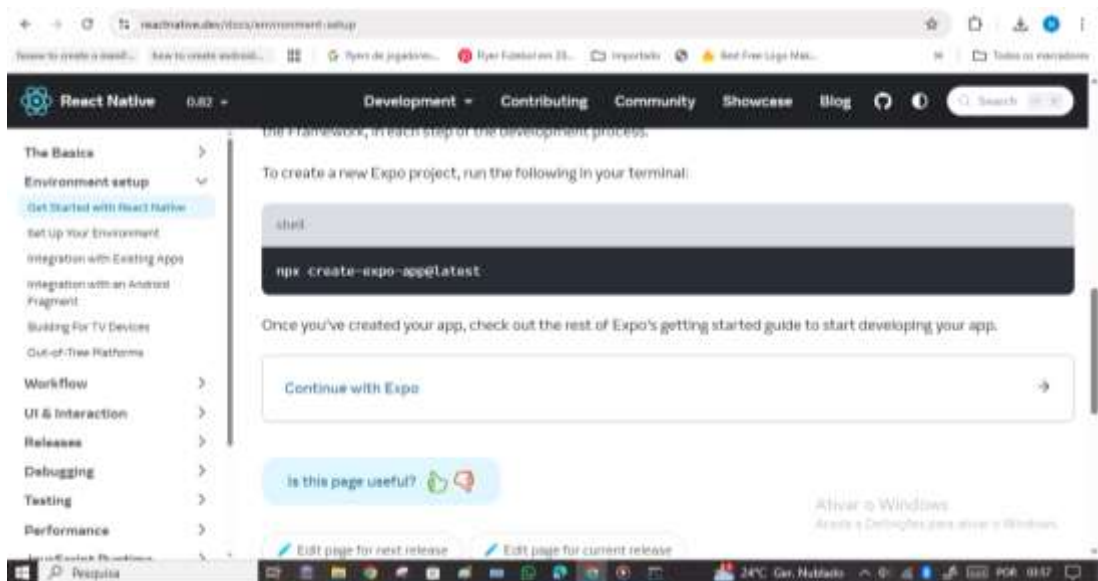


Figura 10 - Site oficial react native | comando para criação de projecto react

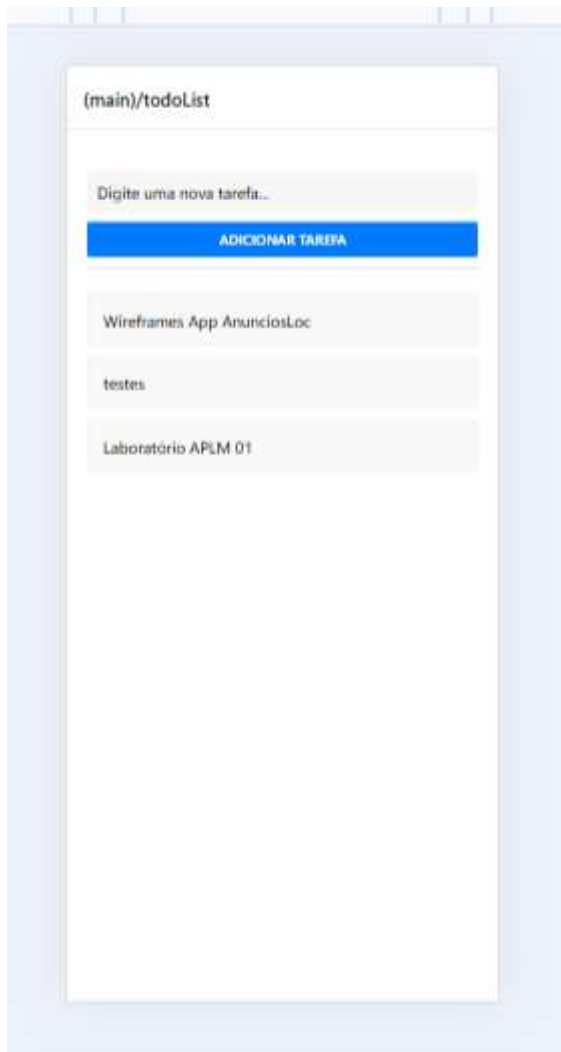


Figura 11 - Lista d tarefas em Execução (Exercício III)

Observações – Build e Dependências

O processo de execução (`expo start`) é vulnerável a problemas de rede/Firewall, manifestados pelo `TypeError: fetch failed` durante a busca de metadados. O uso de **TypeScript** (`.tsx`) requer a definição de **Interfaces** (`interface Todo`) para evitar erros de tipagem (`never`) no *state* inicial.

[illegible]

Figura 12 - `TypeError: fetch failed`

3. Desafios

O projeto foi concebido sob a arquitetura **Monorepo** (npm Workspaces) para encapsular a modularidade exigida pelo Exercício 3.3. A execução do *software* é baseada no **paradigma declarativo do React** e utiliza o **Expo Router** para a navegação.

1. Estrutura Modular e Arquitetura

O software é dividido em módulos lógicos, geridos pelo package.json raiz (Monorepo):

| Módulo/Pacote | Depende de: | Função na Estrutura |
|----------------------|------------------------------------|---|
| @monorepo/common | Ninguém | Camada de Domínio: Contém a lógica reutilizável (ex: <code>Quote.js</code>). |
| @monorepo/api | @monorepo/common | Camada de Interface: Define os contratos e modelos de dados usados pela aplicação principal. |
| @monorepo/mobile-app | @monorepo/common, @monorepo/api | Camada de Apresentação: Contém os Componentes React e a UI. É o ponto de entrada (<code>App.js</code>). |

2. Desenho da Interface Móvel (*Wireframe* de Atividade)

O ecrã da aplicação de Lista de Tarefas (Exercício III) consiste num **Componente Funcional Único** (`ToDoListScreen.tsx`), substituindo o conceito de Activity única do Android.

Wireframe da Atividade (ToDoListScreen)

O ecrã é organizado verticalmente, como solicitado no original:

1. **Cabeçalho/Espaçamento (StatusBar):** Garante a segurança visual no topo do dispositivo.
2. **Área de Entrada (<View>):**
 - **Entrada de Texto:** `<TextInput>` – Para o utilizador digitar a nova tarefa.
 - **Ação:** `<Button>` – Aciona o evento de inserção da tarefa.
3. **Lista de Tarefas:** `<FlatList>` – Ocupa o restante ecrã, exibindo os itens.

Explicação Sucinta do Wireframe:

A interface é minimalista, utilizando o layout **Flexbox** (padrão React Native) para organizar o campo de inserção e o botão numa área superior, enquanto o componente `<FlatList>` é dimensionado para preencher o espaço restante (flex: 1). A conceção é orientada ao **componente**, onde o `ToDoListScreen` encapsula toda a UI e lógica, sem depender de `Activities` ou XML de *layout* externos.

3. Detalhes de Implementação (Arquitetura e Estruturas de Dados)

Arquitetura

- **Ponto de Entrada:** `index.tsx` (Expo Router) -> Lógica de Redirect para a rota principal.
- **Fluxo de Controlo:** Gerido pelo **Expo Router**, usando o `router.push` para navegar no *stack* entre os ecrãs.
- **Compilação:** O **Metro Bundler** é o sistema de *build* ativo, transformando o código TypeScript/JSX em JavaScript compatível.
- **Controlo de Estado:** A gestão de *state* da aplicação é local, utilizando *hooks* React.

Estruturas de Dados

A estrutura de dados para a Lista de Tarefas é um *array* de objetos tipados, garantindo a rastreabilidade e a capacidade de identificação única dos itens:

TypeScript

```
// Interface em TypeScript
interface Todo {
  key: string; // Chave única para o FlatList (substitui o Adapter ID)
  text: string; // O conteúdo da tarefa
}

// Implementação no State
const [todos, setTodos] = useState<Todo[]>([]);
```

Esta seção fornece uma visão panorâmica da arquitetura e da estrutura do software concebido, especificamente no **contexto adaptado (React Native/Expo)**, conforme solicitado, detalhando o *wireframe* e os elementos técnicos de implementação.

Panorâmica da Estrutura do Software (Contexto Adaptado: React Native/Expo)

O projeto foi concebido sob a arquitetura **Monorepo** (npm Workspaces) para encapsular a modularidade exigida pelo Exercício 3.3. A execução do *software* é baseada no **paradigma declarativo do React** e utiliza o **Expo Router** para a navegação.

1. Estrutura Modular e Arquitetura

O *software* é dividido em módulos lógicos, geridos pelo **package.json raiz** (Monorepo):

| Módulo/Pacote | Depende de: | Função na Estrutura |
|----------------------|------------------------------------|--|
| @monorepo/common | Ninguém | Camada de Domínio: Contém a lógica reutilizável (ex: Quote.js). |
| @monorepo/api | @monorepo/common | Camada de Interface: Define os contratos e modelos de dados usados pela aplicação principal. |
| @monorepo/mobile-app | @monorepo/common, @monorepo/api | Camada de Apresentação: Contém os Componentes React e a UI. É o ponto de entrada (App.js). |

Exportar para as Planilhas

2. Desenho da Interface Móvel (*Wireframe* de Atividade)

O ecrã da aplicação de Lista de Tarefas (Exercício III) consiste num **Componente Funcional Único** (ToDoListScreen.tsx), substituindo o conceito de Activity única do Android.

Wireframe da Atividade (ToDoListScreen)

O ecrã é organizado verticalmente, como solicitado no original:

1. **Cabeçalho/Espaçamento (StatusBar):** Garante a segurança visual no topo do dispositivo.
2. **Área de Entrada (<View>):**
 - **Entrada de Texto:** <TextInput> – Para o utilizador digitar a nova tarefa.
 - **Ação:** <Button> – Aciona o evento de inserção da tarefa.
3. **Lista de Tarefas:** <FlatList> – Ocupa o restante ecrã, exibindo os itens.

Explicação Sucinta do Wireframe:

A interface é minimalista, utilizando o layout **Flexbox** (padrão React Native) para organizar o campo de inserção e o botão numa área superior, enquanto o componente <FlatList> é dimensionado para preencher o espaço restante (flex: 1). A conceção é orientada ao **componente**, onde o ToDoListScreen encapsula toda a UI e lógica, sem depender de Activities ou XML de *layout* externos.

3. Detalhes de Implementação (Arquitetura e Estruturas de Dados)

Arquitetura

- **Ponto de Entrada:** `index.tsx` (Expo Router) -> Lógica de Redirect para a rota principal.
- **Fluxo de Controle:** Gerido pelo **Expo Router**, usando o `router.push` para navegar no *stack* entre os ecrãs.
- **Compilação:** O **Metro Bundler** é o sistema de *build* ativo, transformando o código TypeScript/JSX em JavaScript compatível.
- **Controlo de Estado:** A gestão de *state* da aplicação é local, utilizando *hooks* React.

Estruturas de Dados

A estrutura de dados para a Lista de Tarefas é um *array* de objetos tipados, garantindo a rastreabilidade e a capacidade de identificação única dos itens:

TypeScript

```
// Interface em TypeScript
interface Todo {
  key: string; // Chave única para o FlatList (substitui o Adapter ID)
  text: string; // O conteúdo da tarefa
}

// Implementação no State
const [todos, setTodos] = useState<Todo[]>([]);
```

Detalhes Técnicos Chave

Detalhe Técnico Explicação

| | |
|-----------------------|---|
| Tipagem | Uso de TypeScript (<code>.tsx</code>) com Interfaces (<code>interface Todo</code>) para tipar explicitamente o <code>useState<Todo[]>()</code> , resolvendo o erro de inferência de tipo (<code>never</code>). |
| Reatividade | A atualização da lista é obtida por imutabilidade e reatividade . A função <code>addTask</code> utiliza o operador <i>spread</i> (<code>...todos</code>) para criar uma nova instância do <i>array</i> , notificando o React sobre a mudança e acionando o re-render automático do <code><FlatList></code> . |
| Posicionamento | O posicionamento do novo item no topo da lista é assegurado pela sintaxe <code>setTodos([newItem, ...todos])</code> . |

Detalhe Técnico Explicação

Execução O *build* é executado via `npm start` e a *runtime* é o **JavaScript Core** do dispositivo móvel (via Expo Go), em vez da Android Runtime (ART).

4. Referências Bibliográficas

- (MDN), M. (s.d.). Fonte: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/return>
- Microsoft. (2025). *Interfaces*. Fonte: <https://www.typescriptlang.org/docs/handbook/interfaces.html>
- npm, I. /. (2025). *docs.npmjs.com*. Fonte: docs.npmjs.com: <https://docs.npmjs.com/cli/v10/using-npm/workspaces>
- OpenJS Foundation / npm, I. (s.d.). *npm Docs*. Fonte: docs.npmjs.com: <https://docs.npmjs.com/>
- RN, M. /. (2025). *components-and-apis*. Fonte: reactnative.dev: <https://reactnative.dev/docs/components-and-apis>
- RN, M. /. (2025). *flatlist*. Fonte: reactnative.dev: <https://reactnative.dev/docs/flatlist>
- RN, M. /. (2025). *metro*. Fonte: reactnative.dev: <https://reactnative.dev/docs/metro>
- Team, E. (2025). *expo*. Fonte: docs.expo.dev: <https://docs.expo.dev/>
- Team, E. (2025). *introduction*. Fonte: docs.expo.dev: <https://docs.expo.dev/router/introduction/>
- Team, M. /. (2025). *useState*. Fonte: react.dev/: <https://react.dev/reference/react/useState>

5. Repositório GitHub

https://github.com/OlegAnt12/APLM2025_26-L01-Android.git