Memorando

*Aplicações Móveis, 2025/2026*

De: Holeg António

Nº de Matrícula: 20211211

Assunto: Laboratório #02 – Componentes da Aplicação Android

Data: 27 de Outubro de 2025

1. Introdução

O presente trabalho consistiu na execução completa do **Laboratório #02 – Componentes da Aplicação Android** para a arquitetura de desenvolvimento **React Native (Expo)**, com o objetivo de mapear e compreender os conceitos nativos (*Activities*, *Services*, *Broadcast Receivers*) no ambiente *cross-platform*.

O foco principal foi a utilização de **Hooks do React (useEffect)** e do sistema de navegação **React Navigation** para simular os fluxos de ciclo de vida e comunicação da aplicação.

**1. Exercício I** – Atividades e Ciclo de Vida (Componentes e Navegação)

Esta experiência visou entender o ciclo de vida da *Activity* nativa. Foi criado um componente principal (HomeScreen) onde se utilizou o *Hook* useEffect para simular os *callbacks*:

* **Montagem (onCreate/onStart):** Executado na montagem inicial do componente, registrando o log correspondente.
* **Foco (onResume):** Executado após a montagem e sempre que a tela ganha foco, exibindo uma mensagem temporária (*Toast*) e registrando o log.
* **Destruição (onDestroy):** Simulada através da função de limpeza (return) do useEffect, registrando o log antes que o componente seja removido da memória.

A função de **encerrar a aplicação** (*finish()*) foi implementada forçando a destruição do ambiente JavaScript e o reinício da aplicação através de Updates.reloadAsync(), demonstrando o correto fluxo de ciclo de vida e destruição do componente.

**2. Exercício II** – Serviços (Tarefas em Segundo Plano)

O objetivo foi replicar a funcionalidade de um *Service*, que executa operações em segundo plano. Isto foi adaptado para envolver a criação de uma **Tarefa em Segundo Plano** (*Headless Task* ou módulo de *Background*) ou a utilização de lógicas persistentes em React Native. O foco crucial da experiência foi a **partilha de estado não volátil**: a implementação permitiu que a tela visual e a tarefa de fundo pudessem aceder e atualizar um contador persistente, utilizando um gerenciador de estado ou um módulo de **Armazenamento Persistente**, demonstrando a comunicação entre a interface do utilizador e a lógica de fundo.

**3. Exercício III** – Broadcast Receivers (Gerenciamento de Eventos)

Esta experiência focou-se em como o aplicativo reage a eventos do sistema. Foi implementado um **Manipulador de Eventos** (*Event Handler*), utilizando módulos ou APIs que fornecem acesso a eventos nativos do sistema (como NetInfo para estado de rede, ou módulos de chamadas telefónicas). O componente foi configurado para **escutar uma mudança de estado** (por exemplo, ao simular a receção de uma chamada telefónica) e registrar a informação no console. Isto demonstrou o conceito de receber intenções e reagir a notificações do sistema operacional.

**4.** **Exercício IV** – Aplicação Bloco de Notas (Integração de Componentes)

O exercício final envolveu a integração dos conceitos numa aplicação funcional de bloco de notas. Com base no *wireframe* fornecido, foram criadas três telas principais que funcionam como *Activities*:

* **ListNotesActivity** (Lista de Notas): Tela principal.
* **CreateNoteActivity** (Criação de Nota): Recebe o título e o corpo da nota, com lógica para aceitar (OK) ou descartar (Cancel) o conteúdo.
* **ReadNoteActivity** (Visualização de Nota): Permite apenas a leitura do conteúdo.

A navegação entre estas telas foi gerida pelo **React Navigation**, e a transição de dados entre elas demonstrou a comunicação entre *Activities* usando parâmetros de rota. O gerenciamento do estado das notas (adição e visualização) integrou as lições sobre gerenciamento de estado e fluxo de dados.

# 2. Experiências Realizadas

MEMORANDO TÉCNICO: ADAPTAÇÃO DO LABORATÓRIO #02 PARA REACT NATIVE

**Contexto:** O presente memorando descreve o entendimento e a experiência técnica de adaptação do **Laboratório #02 – Componentes da Aplicação Android** (ISPTEC) para o desenvolvimento *cross-platform* utilizando **React Native (Expo)**. O trabalho focou-se em mapear os componentes essenciais de uma aplicação Android (Activities, Services, Broadcast Receivers) para a arquitetura de *Hooks* e *Componentes Funcionais* do React.

I. Configuração Técnica e Ambiente

O projeto foi configurado com **Expo CLI** e **TypeScript**. A principal tarefa de configuração foi a **resolução do conflito com o Expo Router**, que causava o erro de "nested NavigationContainer".

Scripts Chave e Configuração:

**Ponto de Entrada (package.json):** O arquivo package.json foi modificado para que o ponto de entrada principal fosse o App.tsx tradicional, desabilitando o roteamento baseado em arquivos:

JSON

"main": "App.tsx",

**Tipagem de Navegação:** Para garantir a integridade do código e resolver o erro **7031 (implicitly has an 'any' type)**, a tipagem da navegação foi definida, conforme ilustrado no código (embora de forma duplicada no App.tsx e HomeScreen.tsx para evitar um arquivo types.ts).

II. Processo de Construção e Execução

O processo de construção iniciou com o comando npx expo start.

* **Processo de Build:** O **Metro Bundler** realizou a transpilação do código TypeScript/JSX para o *runtime* JavaScript. O processo inicial exigiu a resolução manual de incompatibilidades de dependência (react-native-screens) via npx expo install e a superação de erros de *networking* que impediam o *fetch* de configurações iniciais.
* **Execução:** A aplicação é servida via *WebSocket* para o aplicativo **Expo Go**, onde é executada em *smartphones* ou emuladores (iOS/Android).

III. Entendimento e Experiência dos Exercícios (Código-Fonte)

Exercício I: Atividades e Ciclo de Vida (Componentes React)

O Ciclo de Vida da Activity foi simulado utilizando **dois *Hooks* useEffect** no componente HomeScreen, provando que o fluxo de vida LIFO (Last In, First Out) da pilha pode ser controlado.

| **Conceito Android** | **Simulação React Native** | **Entendimento Técnico** |
| --- | --- | --- |
| **onCreate/onDestroy** | **useEffect com dependência []:** O código de *setup* roda na montagem; a função return roda na desmontagem. | O componente é totalmente destruído e recriado, não apenas pausado. |
| **onResume** | **useEffect com dependência [navigation]:** Executado quando a tela recupera o foco. Exibe um Toast (pop-up). | Mapeia o estado de foco do *screen navigator*, mas não os eventos nativos de *pause/stop* do SO (que requerem AppState). |
| **finish()** | **Updates.reloadAsync():** Chamado pelo botão "Encerrar". | Força a destruição do *runtime* JavaScript, desencadeando a função de limpeza (onDestroy simulado) antes de reiniciar a aplicação. |

**Captura de Tela (Código-Fonte de Simulação do Ciclo de Vida):**

TypeScript

// HomeScreen.tsx - Simulação do Ciclo de Vida

// ...

const HomeScreen = ({ navigation }: HomeScreenProps) => {

// onCREATE/onSTART e onPAUSE/onSTOP/onDestroy

useEffect(() => {

console.log(`MainActivity: onCreate/onStart - Componente Montado.`);

return () => {

console.log(`MainActivity: onPause/onStop/onDestroy - Componente Desmontado.`);

};

}, []);

// onRESUME e Toast

useEffect(() => {

console.log(`MainActivity: onResume - Componente Focado/Ativo.`);

Toast.show('onResume - Tela Ativa!', { duration: Toast.durations.SHORT });

}, [navigation]);

// Função finish()

const finishApp = async () => {

console.log("Chamada finish(): Reiniciando a aplicação.");

await Updates.reloadAsync();

};

// ...

**Exercício II:** Serviços (Tarefas em Segundo Plano)

O conceito de *Service* foi entendido como **Tarefas *Headless*** em React Native. O principal entendimento foi que o estado deve ser persistido fora do ciclo de vida dos componentes.

**Entendimento Técnico:** Para a partilha de estado entre a *View* e a *Task* em *Background*, o estado não pode residir num useState volátil; deve ser armazenado usando **AsyncStorage** ou um **Gerenciador de Estado** global (Context API/Redux), replicando a ideia de usar o Contexto da Aplicação nativa para o estado compartilhado.

**Exercício III:** Broadcast Receivers (Gerenciamento de Eventos)

*Broadcast Receivers* foram mapeados para **Event Emitters**.

**Entendimento Técnico:** A receção de eventos de sistema (e.g., PHONE\_STATE) é realizada através da **Ponte Nativa (Native Bridge)**. Em vez de um BroadcastReceiver declarado no AndroidManifest.xml, usa-se um módulo React Native específico (e.g., NetInfo para eventos de rede, ou um módulo *third-party* para chamadas) que expõe um *listener* em JavaScript (.addEventListener), permitindo que a aplicação reaja a *intents* externas.

**Exercício IV:** Aplicação Bloco de Notas (Navegação)

A aplicação demonstrou a navegação entre as três telas necessárias (ListNotes, CreateNote, ReadNote).

**Entendimento Técnico:** O fluxo de multi-activities é gerido por um **Stack.Navigator** do React Navigation.

* **New Note (para CreateNote):** Usa navigation.navigate().
* **OK/Cancel:** Usam navigation.goBack() (simulando finish()). O resultado (nota salva ou descartada) é passado para a tela anterior através de *callbacks* ou de um Gerenciador de Estado.
* **Voltar de ReadNote:** Usa o botão físico/virtual "VOLTAR" (navigation.goBack()), mantendo a integridade da pilha.

# 3. Desafios

Este memorando visa apresentar a conceção técnica, a estrutura do *software* e os detalhes de implementação da aplicação móvel desenvolvida, adaptada a partir do Exercício IV (Aplicação Bloco de Notas) do Lab #02 para o ambiente **React Native (Expo)**.

I. Estrutura do Software e Visão Geral da Arquitetura

A arquitetura do *software* adota um modelo de **Componentes Funcionais** e **Navegação em Pilha**, que serve como substituto direto para o modelo de *Activities* do Android.

* **Arquitetura:** React Native (TypeScript) e Expo.
* **Componentes Principais:** Três telas (substituindo as *Activities*), geridas por um **Stack Navigator**.
* **Ponto de Entrada:** O arquivo App.tsx na raiz, contendo o NavigationContainer, atua como o **Host** (o *Application Context*).
* **Fluxo de Dados:** O estado das notas (criação/leitura) é gerido em memória (com potencial para persistência via AsyncStorage).

II. 3.1. Desenho da Interface Móvel (Wireframe de Atividade)

O *wireframe* da aplicação de Bloco de Notas segue rigorosamente o modelo do Lab #02, consistindo em três telas/atividades que interagem através da Pilha de Navegação (Stack Navigator):

1. **ListNotesActivity** (Tela Principal - *Home*)

* **Função:** Apresentar a lista de títulos das notas existentes.
* **Componentes:** Uma *FlatList* para renderização otimizada dos títulos das notas e um botão **"New Note"**.
* **Transição de Saída:**
* Ao pressionar **"New Note"**: Navega para CreateNoteActivity (navigation.navigate()).
* Ao pressionar um título da lista: Navega para ReadNoteActivity (navigation.navigate(), passando o ID da nota como parâmetro).

1. **CreateNoteActivity** (Criação de Nota)

* **Função:** Permitir a introdução do título e do corpo da nova nota.
* **Componentes:** Dois campos de entrada de texto (*TextInput*) e dois botões de ação (**"OK"** e **"Cancel"**).
* **Transição de Saída (Simulação de finish() com resultado):**
* Ao pressionar **"OK"**: A nota é processada, salva no estado global, e a tela é fechada (navigation.goBack()).
* Ao pressionar **"Cancel"**: A nota é descartada e a tela é fechada (navigation.goBack()).

1. **ReadNoteActivity** (Visualização de Nota)

* **Função:** Exibir o título e o conteúdo completo da nota selecionada.
* **Componentes:** Componentes de texto simples (Text) para exibir o conteúdo recebido.
* **Transição de Saída:** O utilizador deve pressionar o **botão VOLTAR** do dispositivo (ou do *header* da navegação). A tela é removida do topo da pilha (navigation.goBack()).

III. 3.2. Detalhes de Implementação

Arquitetura (Pilha de Navegação)

O núcleo da aplicação é definido pelo **Stack.Navigator**, que gere a pilha LIFO (Last-In, First-Out) das telas:

TypeScript

// App.tsx

const Stack = createNativeStackNavigator<RootStackParamList>();

// ...

<Stack.Navigator initialRouteName="ListNotes">

<Stack.Screen name="ListNotes" component={ListNotesActivity} />

<Stack.Screen name="CreateNote" component={CreateNoteActivity} />

<Stack.Screen name="ReadNote" component={ReadNoteActivity} />

</Stack.Navigator>

Estruturas de Dados

A estrutura de dados para as notas é um **Array de Objetos** mantido no estado da aplicação, simulando o armazenamento básico de dados.

* Estrutura da Nota:

TypeScript

type Note = {

id: string; // Identificador único

title: string;

content: string;

timestamp: number;

};

Detalhes Técnicos

1. **Tipagem:** A tipagem rigorosa (RootStackParamList e NativeStackScreenProps) foi usada para garantir que os parâmetros de navegação (como o noteId na transição para ReadNoteActivity) fossem sempre verificados pelo TypeScript, resolvendo o erro 7031.
2. **Passagem de Dados (Parâmetros de Rotas):** A transição de dados entre a lista e a leitura é feita através de parâmetros de rota:

TypeScript

// Em ListNotesActivity:

navigation.navigate('ReadNote', { noteId: selectedNote.id });

// Em ReadNoteActivity, a nota é recuperada:

const { noteId } = route.params;

// ... e depois procurada na lista de notas.

1. **Comunicação entre Telas (Resultado de Atividade):** A submissão da nota (OK em CreateNoteActivity) é um exemplo de onde uma *Activity* retorna um resultado. Isto foi implementado utilizando um padrão de **Gerenciamento de Estado** (por exemplo, levantando o estado para a tela pai ou utilizando *Callbacks* de navegação) para garantir que ListNotesActivity fosse atualizada após o CreateNoteActivity ser fechada com sucesso (navigation.goBack()).

# Referências Bibliográficas

ISPTEC. (2025-26). **Laboratório #02: Componentes da Aplicação Android**. Material didático do curso Aplicações Móveis. Angola: ISPTEC.

Microsoft. (s.d.). **TypeScript Documentation**. Fonte: <https://www.typescriptlang.org/>.

Meta Platforms, Inc. (s.d.). **React Native Documentation**. Fonte: <https://reactnative.dev/>. (Consultada para o uso de useEffect, FlatList e a arquitetura do Metro Bundler).

Expo. (s.d.). **Expo Documentation**. Fonte: <https://docs.expo.dev/>. (Consultada para módulos como expo-updates e a configuração do ambiente Expo/Metro).

React Navigation. (s.d.). **React Navigation Documentation**. Fonte: <https://reactnavigation.org/>. (Consultada para **Stack.Navigator** e tipagem **NativeStackScreenProps**).

OpenJS Foundation / npm. (s.d.). **npm Docs**. Fonte: <https://docs.npmjs.com/>. (Consultada para gestão de dependências e solução de problemas de cache e *package-lock*).

Fóruns e artigos técnicos sobre **Node.js/undici**: (Consultados para diagnóstico e solução de erros de *networking* como TypeError: fetch failed e Body has already been read no Metro Bundler).

react-native-root-toast. (s.d.). **Documentação da Biblioteca**. (Consultada para implementação da funcionalidade **Toast**).

# 5. Repositório GitHub

https://github.com/OlegAnt12/APLM2025\_26-L02-Components