

# AULA 03 – Estrutura de Pastas e Arquivos em Projetos React Native com Expo

**Disciplina:** Programação Mobile / Desenvolvimento Front-end

**Objetivo da Aula:**

- Compreender a função das pastas e arquivos essenciais em um projeto React Native com Expo.
- Entender como organizar o código de forma escalável e profissional.

## 1. Introdução

Ao desenvolver aplicações React Native utilizando o Expo, é fundamental compreender como a estrutura de pastas é organizada.

Essa estrutura garante:

- **Escalabilidade** do projeto
- **Padronização** entre equipes
- **Manutenção facilitada**
- **Separação de responsabilidades**

A estrutura que você apresentou é **muito comum em aplicações profissionais**, especialmente quando o projeto começa a crescer.

## 2. Visão Geral da Estrutura

A árvore do projeto vista no VS Code é semelhante a esta:

```
helloWord/
  ├── .expo/
  ├── assets/
  ├── document/
  ├── node_modules/
  └── src/
    ├── assets/
    ├── components/
    ├── hooks/
    ├── navigation/
    └── screens/
```

```
└── services/
    └── utils/
    .gitignore
    App.js
    app.json
    index.js
    LICENSE
    package.json
    package-lock.json
    README.md
    Nota Aula 1.pdf
    Nota Aula 2.pdf
```

Agora vamos examinar cada item.

## 3. Explicação das Pastas Raiz

### 3.1 .expo/

Criada automaticamente pelo Expo.

Armazena configurações internas, cache e metadados usados pelo ambiente Expo.

**Nunca é editada manualmente.**

### 3.2 assets/ (nível raiz)

Contém recursos que são carregados antes da aplicação subir:

- Imagens ícones
- Splash screen
- Fontes locais

O Expo utiliza essa pasta para recursos estáticos essenciais.

### 3.3 document/

Pasta personalizada pelo usuário.

Pode ser usada para:

- Documentação interna
- PDFs (como "Nota de Aula 1" e "Nota de Aula 2")

- Arquivos auxiliares do projeto

Não faz parte da estrutura padrão do Expo.

## 3.4 **node\_modules/**

Diretório onde ficam todas as dependências instaladas via npm.

Exemplo:

- React
- React Native
- Expo
- Bibliotecas adicionais

**Não deve ser editado manualmente.**

# 4. A pasta **src/**: O Coração do Projeto

A pasta **src/** concentra todo o código da sua aplicação.

É uma prática recomendada separar o código principal aqui para manter o projeto organizado.

## 4.1 **src/assets/**

Armazena imagens, fontes e arquivos usados **dentro da aplicação**, mas não os de inicialização.

Exemplos:

- Imagens de botões
- Ilustrações internas
- Fontes customizadas

## 4.2 **src/components/**

Contém componentes reutilizáveis, como:

- Botões
- Cards
- Inputs

- Headers
- Rodapés

**Exemplo:**

Um botão estilizado que será usado em várias telas.

Criar componentes reutilizáveis ajuda a:

- Reduzir duplicação
- Padronizar visual
- Acelerar desenvolvimento

## 4.3 src/hooks/

Armazena **hooks personalizados**, como:

- Lógica de autenticação
- Lógica de requisições
- Controle de formulários
- Estados globais personalizados

Exemplo:

```
useAuth()  
useFetch()  
A utilidade é separar lógica da interface.
```

## 4.4 src/navigation/

Armazena toda lógica de navegação entre telas:

- StackNavigator
- BottomTabNavigator
- DrawerNavigator

Organiza rotas da aplicação, tornando claro como o usuário navega entre telas.

## 4.5 src/screens/

Contém as **telas completas** do aplicativo.

Exemplos:

- HomeScreen
- LoginScreen
- ProfileScreen

Cada tela é um componente principal que representa uma página da interface.

## 4.6 **src/services/**

Aqui ficam os arquivos responsáveis por:

- Comunicação com APIs
- Regras de negócio
- Acesso a banco local (SQLite ou AsyncStorage)

Exemplo:

```
api.js  
authService.js  
userService.js
```

Separar serviços melhora a organização da **lógica de dados**.

## 4.7 **src/utils/**

Funções utilitárias que podem ser usadas em todo o projeto, como:

- Formatação de datas
- Máscaras de input
- Funções matemáticas
- Validações gerais

São funções independentes e reutilizáveis.

# 5. Arquivos importantes no nível raiz

## 5.1 **.gitignore**

Define quais arquivos/pastas o Git deve ignorar.  
Exemplo: node\_modules, arquivos temporários, logs.

## 5.2 App.js

Arquivo principal da aplicação Expo.  
É o primeiro código JavaScript carregado quando o app inicia.

Geralmente contém:

- Providers
- Navegação inicial
- Componentes globais

## 5.3 app.json

Arquivo de configuração do Expo.

Define:

- Nome do app
- Ícone
- Splash screen
- Permissões
- Configurações de build

É o equivalente ao "manifesto" do Expo.

## 5.4 index.js

Ponto de entrada do React Native (camada interna), chamando o App.js.  
No Expo, raramente é alterado.

## 5.5 package.json

Arquivo mais importante do projeto Node.

Contém:

- Nome do projeto

- Scripts (`npm start`, `npm run android` etc.)
- Dependências do projeto
- Versão do app
- Configuração do npm

## 5.6 package-lock.json

Armazena o mapeamento exato das versões instaladas.  
Garante que outros desenvolvedores terão o mesmo ambiente.

## 5.7 README.md

Documento que explica:

- Objetivo do projeto
- Como instalar
- Como rodar
- Stack utilizada

Muito importante para projetos profissionais.

## 5.8 Arquivos PDF (Notas de Aula)

Foram adicionados por você manualmente.  
Servem como documentação auxiliar.

# 6. Por que usar essa organização?

Essa estrutura é essencial em qualquer projeto profissional porque:

### ✓ Facilita manutenção

Mudar um componente ou tela fica simples e rápido.

### ✓ Permite trabalho em equipe

Cada parte do código tem o seu lugar.

### ✓ Escala bem

À medida que o app cresce, a organização evita confusão.

## ✓ Segue boas práticas da comunidade

React Native + Expo → estrutura modularizada.

# 7. Resumo

Pasta/Arquivo	Função
.expo/	Configurações internas do Expo
assets/	Imagens/fontes estáticas iniciais
src/	Todo o código principal da aplicação
components/	Componentes reutilizáveis
screens/	Telas completas do app
navigation/	Sistema de rotas
hooks/	Lógica avançada reutilizável
services/	Acesso a API e regras de negócio
utils/	Funções de apoio
package.json	Configurações do projeto
App.js	Arquivo principal da aplicação
PDFs/document	Materiais auxiliares