Desenvolvimento de Aplicativo Mobile

Material base desenvolvido pelo
Prof. Raphael Barreto

raphael.b.oliveira@docente.senai.br

2026
SENAI - SAPUCAÍ

Gestos na Touch Screen e Animações

CÓDIGO 13 | 2026

Firjan SENAI - Desenvolvimento Mobile com React Native

Sumário

01

Introdução aos Gestos Touch

Diferenças entre interação web e mobile

02

Configuração do Ambiente

Preparando o projeto para gestos

03

Implementação do Fling Gesture

Criando o gesto de arrastar

04

Animações com React Native

Feedback visual para interações

05

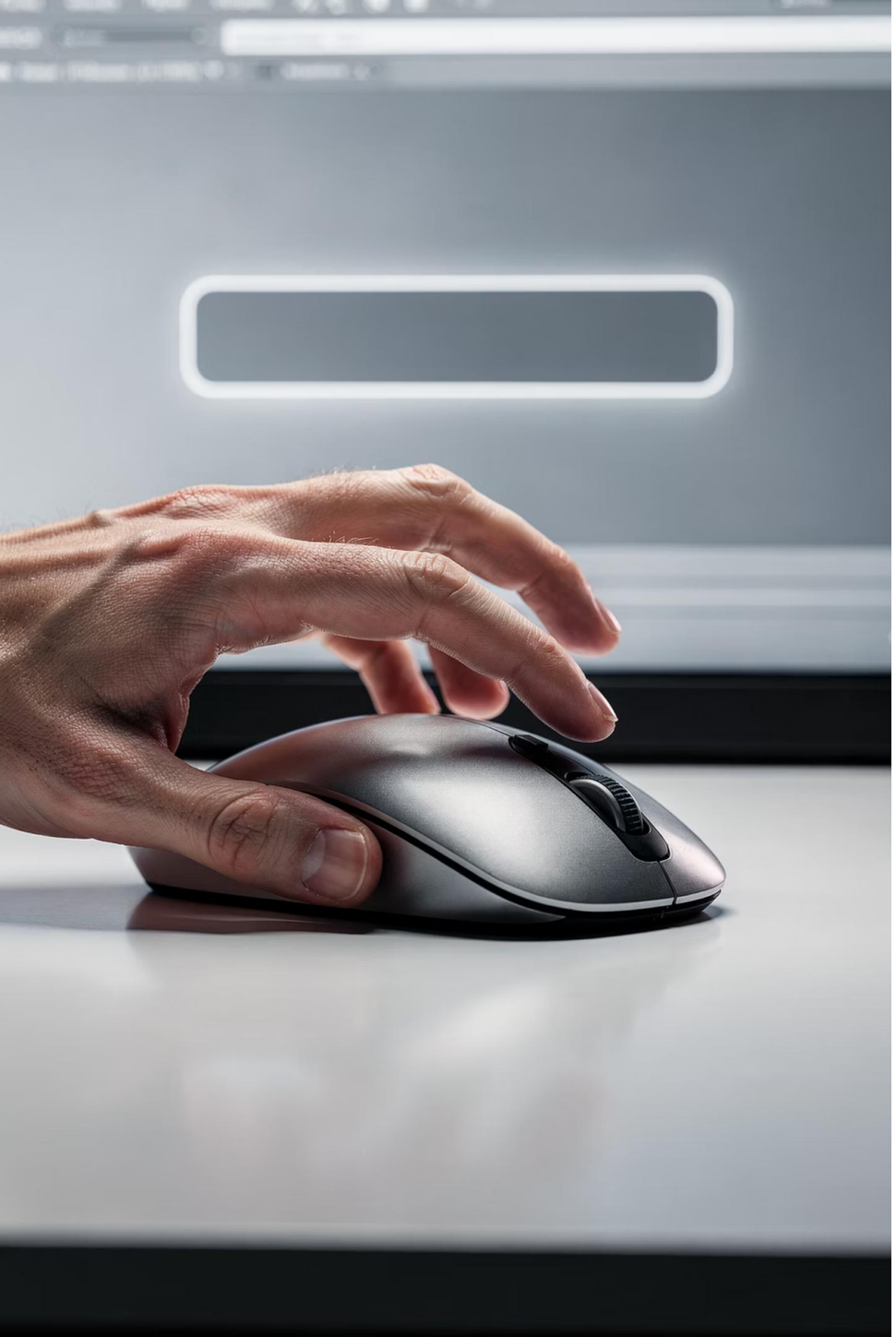
Código Completo

Integração final e boas práticas

01

Introdução aos Gestos Touch

Compreendendo as diferenças fundamentais entre interação web e mobile



Interação Web vs Mobile

Ambiente Web

- Mouse como dispositivo principal
- Evento hover ao passar sobre elementos
- Feedback visual por proximidade
- Telas geralmente não-touch

Ambiente Mobile

- Dedos como interface direta
- Sem evento hover disponível
- Interação por toque direto
- Telas touchscreen padrão

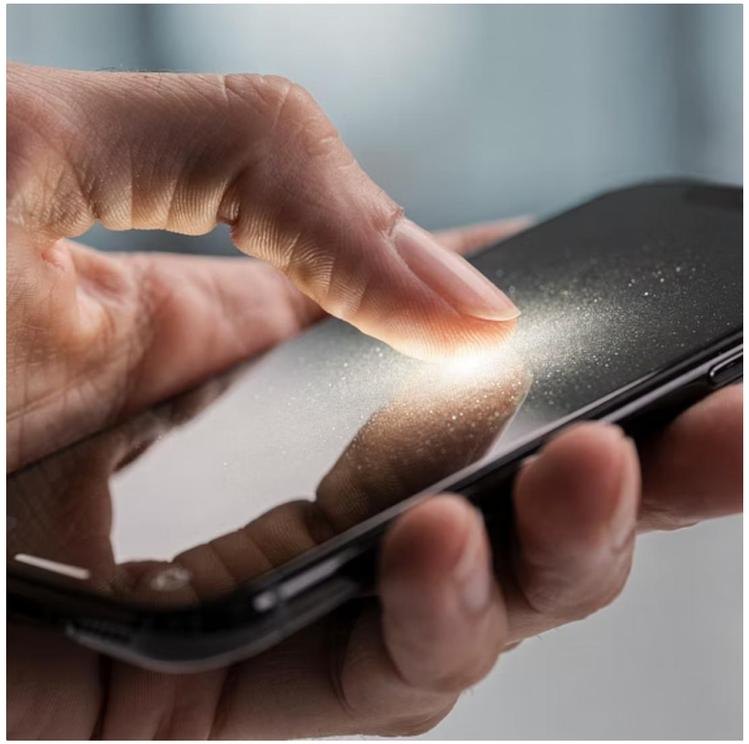
O Comportamento Hover na Web

No desenvolvimento web, o evento hover é fundamental para a experiência do usuário. Quando passamos o mouse sobre um botão clicável, esperamos ver uma mudança visual — uma alteração de cor, um sublinhado, ou qualquer feedback que indique que aquele elemento é interativo.

Este comportamento se tornou tão natural no uso de computadores que usuários inconscientemente buscam essas pistas visuais antes de clicar. No entanto, no ambiente mobile, essa interação simplesmente não existe.

Importante: A ausência do hover em dispositivos móveis exige que repensemos completamente a forma como sinalizamos elementos interativos para o usuário.

Vantagens da Interação Touch



Gestos Naturais

Movimentos intuitivos como pinça para zoom, que não têm equivalente direto no mouse

Swipe e Fling

Arrastar elementos rapidamente para navegar ou executar ações específicas

Multi-touch

Capacidade de usar múltiplos dedos simultaneamente para interações complexas

Principais Gestos Touch

Tap (Toque)

O gesto mais básico, equivalente ao clique do mouse. Usado para selecionar, ativar botões e interagir com elementos.

Pinch (Pinça)

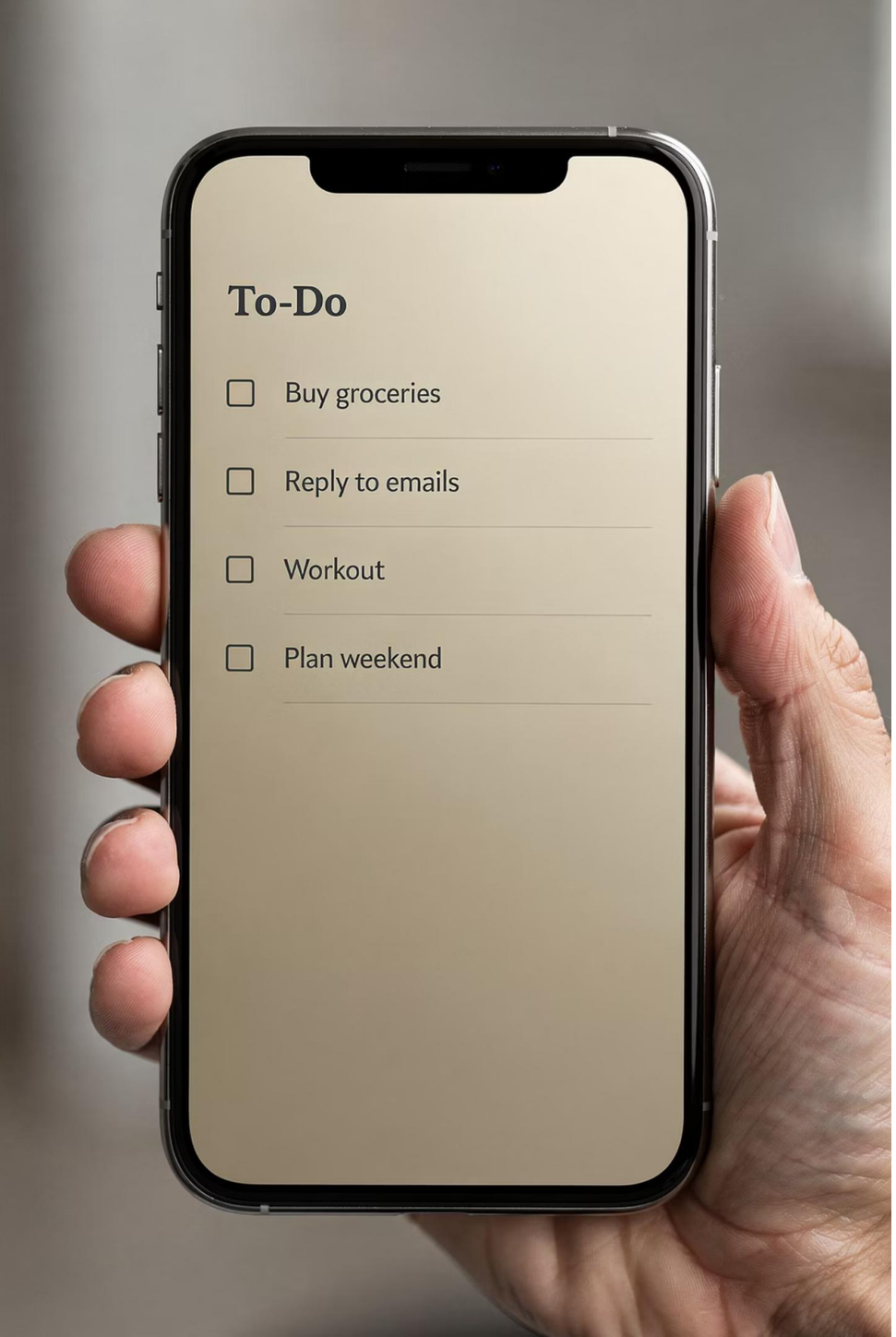
Gesto com dois dedos para aumentar ou diminuir o zoom em imagens, mapas e conteúdo em geral.

Fling (Arremesso)

Movimento rápido de arrastar em uma direção, usado para deslizar conteúdo ou executar ações como deletar.

Pan (Arrastar)

Segurar e mover continuamente um elemento, ideal para reorganizar listas ou navegar em mapas.



Nosso Projeto: Lista de Tarefas

Vamos trabalhar com uma aplicação de lista de tarefas que já possui funcionalidades básicas implementadas. O app permite adicionar novas tarefas e marcar itens como concluídos ou não concluídos.

A estrutura atual conta com dois componentes principais: o layout principal que gerencia toda a lógica da lista, e o componente Task que renderiza cada tarefa individual com sua funcionalidade de marcar como feita.

02

Configuração do Ambiente

Preparando o projeto para reconhecer e processar gestos touch

React Native Gesture Handler

Para trabalhar com gestos avançados em React Native, utilizamos a biblioteca React Native Gesture Handler. Esta biblioteca oferece uma API robusta e performática para reconhecer diferentes tipos de gestos, indo além dos eventos básicos de toque.

Instalação

A biblioteca já vem instalada quando você cria um projeto com Expo. Caso necessite instalá-la manualmente:

```
npx expo install react-native-gesture-handler
```

Vantagens

- Performance otimizada
- Suporte nativo a gestos complexos
- Compatibilidade iOS e Android
- Gestos personalizáveis

GestureHandlerRootView

Antes de implementar qualquer gesto, é necessário envolver toda a aplicação com o componente GestureHandlerRootView. Este componente cria o contexto necessário para que todos os gestos sejam reconhecidos em qualquer parte do app.

> **Atenção: O GestureHandlerRootView deve ser o componente mais externo da sua aplicação. Sem ele, nenhum gesto será detectado corretamente.**

Esta configuração precisa ser feita apenas uma vez no componente raiz (layout principal), permitindo que todos os componentes filhos reconheçam gestos automaticamente.

Configuração Inicial - Parte 1

Vamos começar importando e configurando o GestureHandlerRootView no arquivo principal:

```
import { GestureHandlerRootView } from "react-native-gesture-handler"
```

Esta importação traz o componente que envolverá toda nossa aplicação. Note que importamos diretamente da biblioteca react-native-gesture-handler.

Configuração Inicial - Parte 2

Agora modificamos a estrutura do componente para incluir o GestureHandlerRootView como elemento mais externo:

```
export default function RootLayout() {
  // ... código existente

  return (
    <GestureHandlerRootView>
      <View style={style.mainContainer}>
        {/* Todo o conteúdo da aplicação */}
      </View>
    </GestureHandlerRootView>
  )
}
```

Com esta estrutura, qualquer gesto dentro do aplicativo será detectável.

03

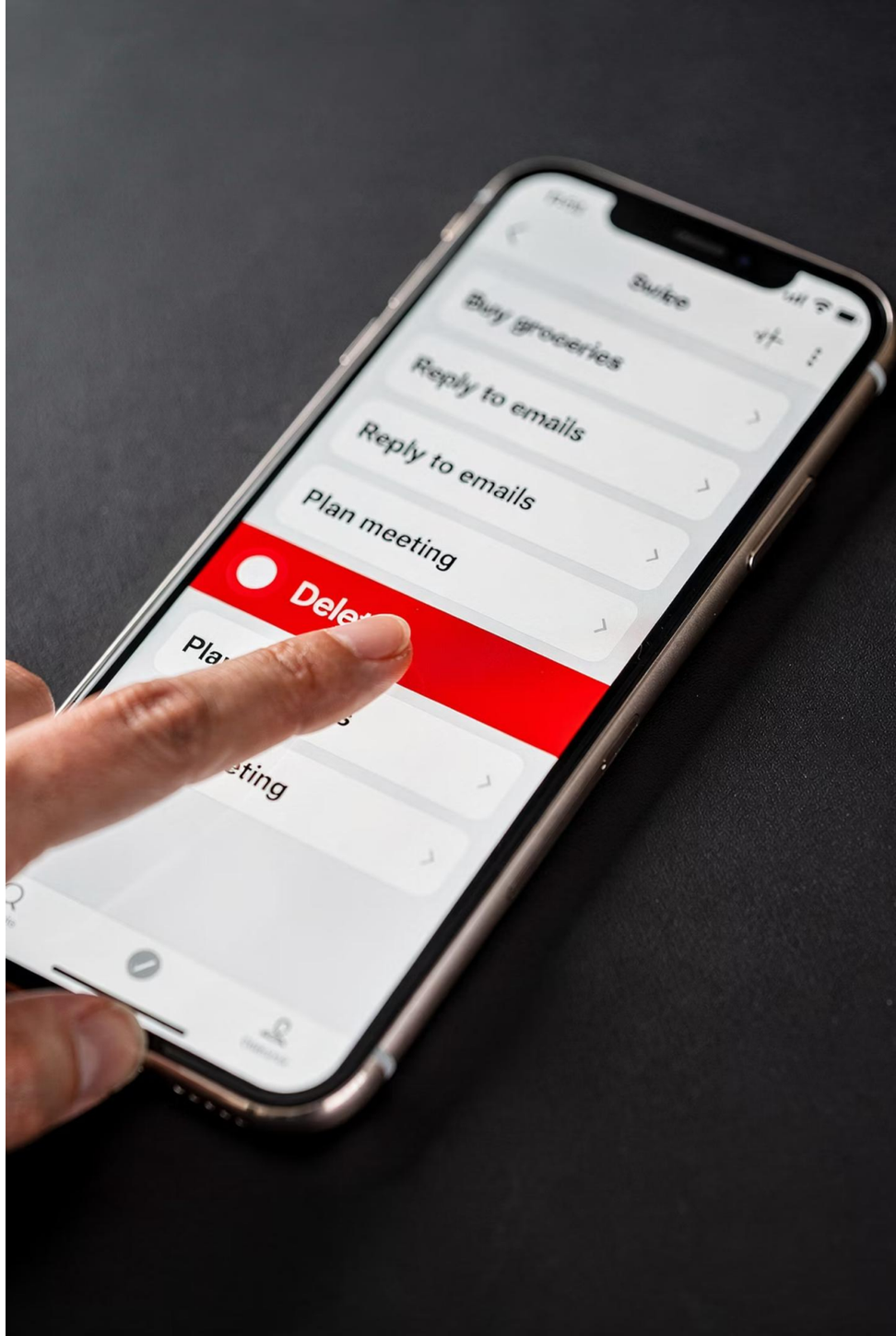
Implementação do Fling Gesture

Criando a funcionalidade de deletar tarefas com gesto de arrastar

Objetivo: Deletar com Swipe

Vamos implementar uma funcionalidade muito comum em aplicativos mobile: deletar um item arrastando-o para o lado. Este padrão de interação é intuitivo para usuários e proporciona uma experiência fluida.

O gesto será um fling para a direita — um movimento rápido de arrastar que, quando executado, removerá a tarefa da lista. Este tipo de interação é familiar para usuários de aplicativos como e-mail e gerenciadores de tarefas.



Criando a Função Delete

Primeiro, implementamos a lógica na FlatList no index para remover uma tarefa da lista.

```
<FlatList
          data={tasks}
          keyExtractor={(item) => item.id}
          renderItem={({ item }) => (
            <Task
              text={item.text}
              initialCompleted={item.completed}
              deleteTask={() => setTasks(tasks.filter((t) => t.id !== item.id))}
            />
          )}
        />
```

Para cada item na lista de tarefas, renderiza um componente Task passando:

- text: o texto da tarefa

- initialCompleted: se a tarefa está concluída ou não

- deleteTask: uma função que deleta a tarefa ao ser chamada com o ID do item

Criando a Função Delete

Antes de implementar o gesto, vamos testar se a função de deletar funciona corretamente. Adicionamos temporariamente um evento de long press no componente Task:

```
export default function Task({ text, initialCompleted, deleteTask }) {
    const [completed, setCompleted] = useState(initialCompleted)
    return (
        <View style={style.rowContainer}>
            <Pressable onPress={() => setCompleted(!completed)}>
                <Ionicons
                    name="checkmark-circle"
                    size={32}
                    color={completed ? colors.primary : "gray"}
                />
            </Pressable>
            <Text onPress={deleteTask}>{text}</Text>
        </View>
    )
}
```

Se ao pressionar e segurar o texto da tarefa ela desaparecer da lista, nossa função está funcionando corretamente. Este teste nos garante que a lógica de remoção está implementada antes de adicionarmos o gesto. Depois do teste, retiramos o onPress do Text e voltamos com o Código anterior.

Importações Necessárias

No componente Task, precisamos importar os elementos para trabalhar com gestos:

```
import { Directions, Gesture, GestureDetector } from "react-native-gesture-handler"
```

- **Directions:** É um objeto (enum) que define as direções cardinais (cima, baixo, esquerda, direita) usadas para configurar gestos de deslize (flings).
- **Gesture**: É o ponto de entrada da biblioteca, utilizado para criar e configurar a lógica de diferentes tipos de gestos (toque, pan, pinça, etc.).
- **GestureDetector:** É o componente visual que envolve os elementos da interface para detectar e reagir aos gestos definidos pelo objeto Gesture.

React Native Manipulador de gestos com Fling gesture

Estrutura do FlingGesture

O FlingGesture envolve o elemento que queremos tornar "arrastável". Veja a estrutura implementada na função Task:

```
export default function Task({ text, initialCompleted, deleteTask }) {
  // Estado para controlar se a tarefa está marcada como concluída ou não
  const [completed, setCompleted] = useState(initialCompleted)

  // Cria uma referência para o valor da animação (começa em 0) que não se perde ao renderizar
  const swipe = useRef(new Animated.Value(0)).current

  // Define a lógica do gesto de “arrasto"
  const flingGesture = Gesture.Fling()
    .direction(Directions.RIGHT) // Configura o gesto para detectar deslize da esquerda para a direita
    .onStart(() => {
      // Quando o gesto é detectado, inicia uma animação de transição
      Animated.timing(swipe, {
        toValue: 500,           // Move o componente 500 pixels para a direita
        duration: 600,          // A animação dura 600 milissegundos
        useNativeDriver: false  // 'false' é necessário aqui pois 'translateX' afeta o layout no Animated padrão
      }).start(() => deleteTask()) // Após o fim da animação, chama a função de exclusão
    })
```

Estrutura do GestureDetector

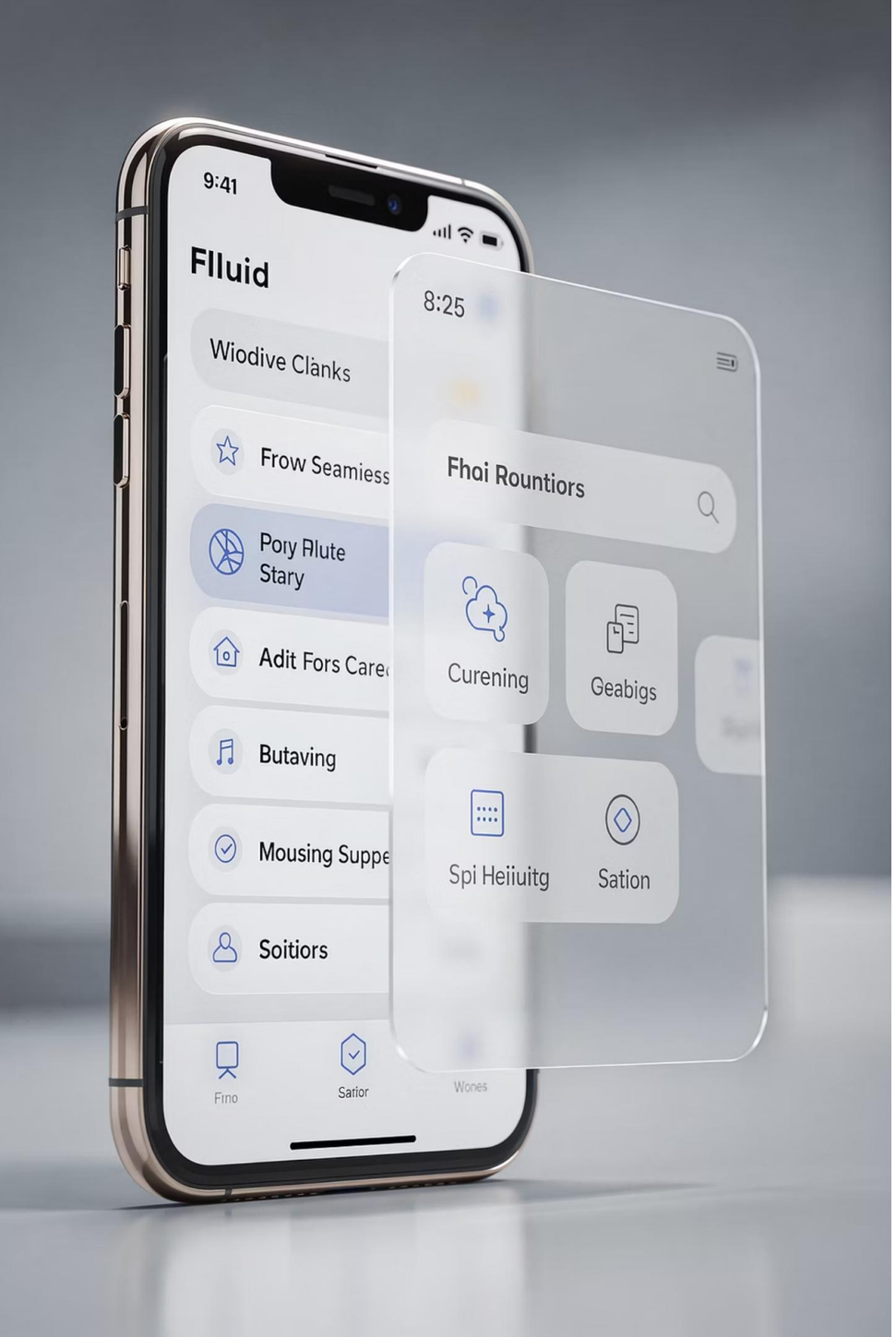
```
return (
    // O GestureDetector é o sensor que "escuta" o flingGesture definido acima
    <GestureDetector gesture={flingGesture}>
      {/* Animated.View permite que o estilo 'transform' seja atualizado pela variável 'swipe' sem re-renderizar
o componente todo */}
      <Animated.View style={[style.rowContainer, { transform: [{ translateX: swipe }] }]}>
        <Ionicons
          name="checkmark-circle"
          size={32}
          color={completed ? colors.primary : "gray"}
          onPress={() => setCompleted(!completed)} // Alterna o estado de conclusão ao tocar no ícone
        />
        <Text>{text}</Text>
      </Animated.View>
    </GestureDetector>
  )
}

...
```

04

Animações com React Native

Adicionando feedback visual às interações do usuário



Por Que Animar?

Gestos e animações geralmente trabalham juntos para criar uma experiência de usuário coesa. Embora seja possível implementar um gesto sem animação, o feedback visual torna a interação mais intuitiva e satisfatória.

Quando o usuário arrasta uma tarefa, ele espera ver aquele movimento acontecendo. A animação cria essa conexão entre ação e resultado, confirmando que o aplicativo está respondendo ao gesto.

React Native Animated API

Sobre a API

O React Native inclui nativamente uma API de animação poderosa e performática. Ela oferece diferentes tipos de animação, sendo as mais comuns:

- timing: animação ao longo do tempo
- spring: animação com efeito de mola
- decay: animação com desaceleração

Vantagens

- Nativa do React Native
- Performance otimizada
- Sem dependências externas
- Integração com Gesture Handler

Preparando a Estilização

Antes de implementar a animação, vamos melhorar visualmente as tarefas para facilitar a visualização do movimento:

```
// Definição dos estilos visuais do componente
const style = StyleSheet.create({
  rowContainer: {
    display: "flex",
    flexDirection: "row",   // Alinha ícone e texto horizontalmente
    alignItems: "center",
    gap: 10,
    marginBottom: 10,
    elevation: 3,           // Sombra para Android
    shadowColor: "#000000", // Configurações de sombra para iOS (shadow...)
    shadowOpacity: 0.1,
    shadowOffset: { width: 0, height: 2 },
    backgroundColor: "#FFFFFF",
    padding: 10,
    borderRadius: 10
  }
})
```

Detalhes de Sombra

elevation (Android)

Propriedade que cria sombra automaticamente no Android. Valor numérico indica a "altura" do elemento.

shadow* (iOS)

Conjunto de propriedades para criar sombra no iOS: shadowColor, shadowOpacity e shadowOffset.

backgroundColor

Essencial para que a sombra apareça corretamente. Sem ela, o fundo fica transparente.

Anatomia do Animated.timing

Transform: A Chave da Animação

A propriedade CSS transform é ideal para animações porque não causa reflow no layout. Suas principais transformações são:

translateX / translateY

Move o elemento horizontalmente ou verticalmente sem afetar outros elementos

rotate

Rotaciona o elemento em graus, útil para indicadores visuais

scale

Aumenta ou diminui o tamanho do elemento proporcionalmente

Perguntas?

Entre em contato:

raphael.b.oliveira@docente.senai.br

Obrigado pela atenção! Estou à disposição para esclarecer dúvidas e ajudar no seu no seu aprendizado.

Firjan SENAI