

INTERFACES WEB II

Pré-requisitos

- Expo Go instalado em um dispositivo físico
- Node.js (versão LTS) instalado
- VS Code ou qualquer outro editor de código ou IDE preferido instalado

Verificar Instalação:

- `node -v`
- `npm -v`

Instalar o Expo CLI

```
npm install -g expo-cli
```

Criando o Projeto:

```
npx create-expo-app stickermash
```


```
cd StickerSmash
```

Baixar Arquivos

- Descompacte o arquivo e substitua os ativos padrão no diretório your-project-name/assets/images .

Executar script reset-project

 terminal

 Cópia

```
- npm run reset-project
```

Execute o aplicativo no celular e na web

 terminal

 Cópia

```
- npx expo start
```

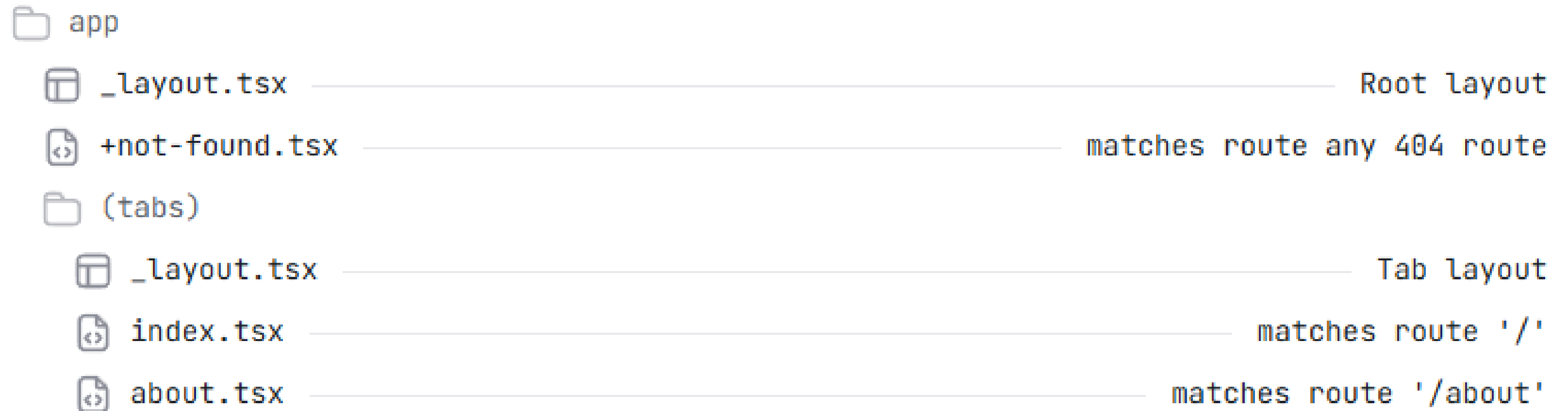
Noções básicas do Expo Router

- O Expo Router é um framework de roteamento baseado em arquivos para React Native e aplicativos web. Ele gerencia a navegação entre telas e utiliza os mesmos componentes em diversas plataformas.

Para começar, precisamos conhecer as seguintes convenções:

- Diretório do aplicativo : um diretório especial contendo apenas rotas e seus layouts. Quaisquer arquivos adicionados a este diretório se tornam uma tela dentro do nosso aplicativo nativo e uma página na web.
- Layout raiz : o arquivo `app/_layout.tsx` . Ele define elementos de interface do usuário compartilhados, como cabeçalhos e barras de guias, para que sejam consistentes entre diferentes rotas.
- Convenções de nome de arquivo : Nomes de arquivos de índice , como `index.tsx` , correspondem ao diretório pai e não adicionam um segmento de caminho. Por exemplo, o arquivo `index.tsx` no diretório do aplicativo corresponde /à rota.
- Um arquivo de rota exporta um componente React como seu valor padrão. Ele pode usar as extensões `.js`, `.jsx`, `.ts` ou `.tsx`.

Estrutura do Projeto:



Adicionar uma rota não encontrada

- Quando uma rota não existe, podemos usá- `+not-found` para exibir uma tela de fallback. Isso é útil quando queremos exibir uma tela personalizada ao navegar para uma rota inválida no celular, em vez de travar o aplicativo ou exibir um erro 404 na web. O Expo Router usa um arquivo especial `+not-found.tsx` para lidar com esse caso.

app/+not-found.tsx

Copy



```
import { View, StyleSheet } from 'react-native';
import { Link, Stack } from 'expo-router';

export default function NotFoundScreen() {
  return (
    <>
      <Stack.Screen options={{ title: 'Oops! Not Found' }} />
      <View style={styles.container}>
        <Link href="/" style={styles.button}>
          Go back to Home screen!
        </Link>
      </View>
    </>
  );
}
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#25292e',
    justifyContent: 'center',
    alignItems: 'center',
  },

  button: {
    fontSize: 20,
    textDecorationLine: 'underline',
    color: '#fff',
  },
});
```

O que é um Stack?

Um navegador de pilha é a base para navegar entre diferentes telas em um aplicativo. No Android, uma rota empilhada é animada sobre a tela atual. No iOS, uma rota empilhada é animada a partir da direita. O Expo Router fornece um componente Stack para criar uma pilha de navegação para adicionar novas rotas

Arquivo de layout Root para adicionar uma (tabs)rota:

app/_layout.tsx

Copy



```
import { Stack } from 'expo-router';

export default function RootLayout() {
  return (
    <Stack>
      <Stack.Screen name="(tabs)" options={{ headerShown: false }} />
    </Stack>
  );
}
```

 app/(tabs)/_layout.tsx

 Copy



```
import { Tabs } from 'expo-router';

export default function TabLayout() {
  return (
    <Tabs>
      <Tabs.Screen name="index" options={{ title: 'Home' }} />
      <Tabs.Screen name="about" options={{ title: 'About' }} />
    </Tabs>
  );
}
```


(tabs)/index.tsx

```
import { Text, View, StyleSheet } from 'react-native';
import { Link } from 'expo-router';

export default function Index() {
  return (
    <View style={styles.container}>
      <Text style={styles.text}>Home screen</Text>
      <Link href="/about" style={styles.button}>
        Go to About screen
      </Link>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#25292e',
    alignItems: 'center',
    justifyContent: 'center',
  },
  text: {
    color: '#fff',
  },
  button: {
    fontSize: 28,
    textDecorationLine: 'underline',
    color: '#fff',
  },
});
```

Adicionar uma nova tela à pilha

```
app/about.tsx  Copy  ⋮

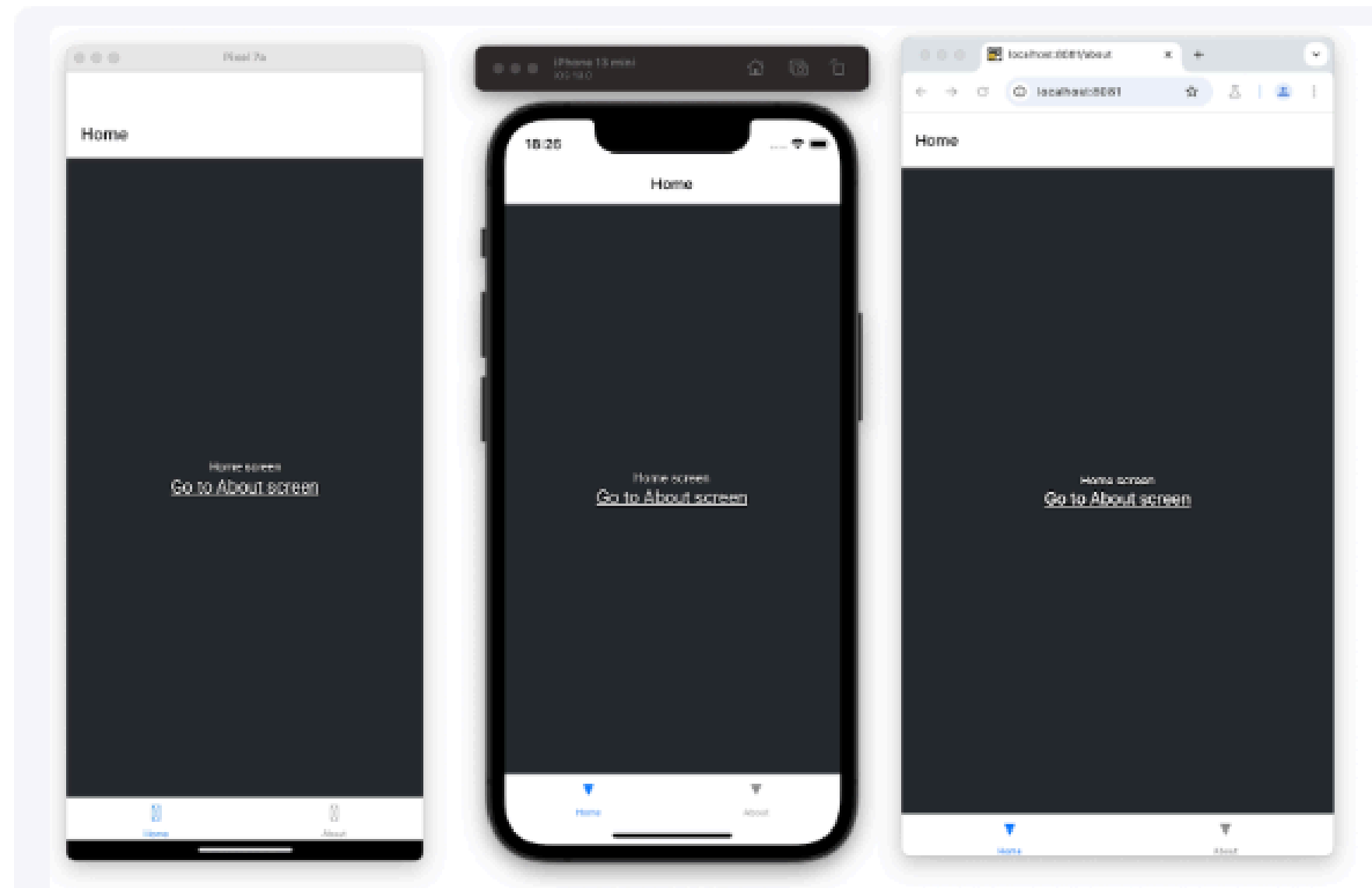
import { Text, View, StyleSheet } from 'react-native';

export default function AboutScreen() {
  return (
    <View style={styles.container}>
      <Text style={styles.text}>About screen</Text>
    </View>
  );
}

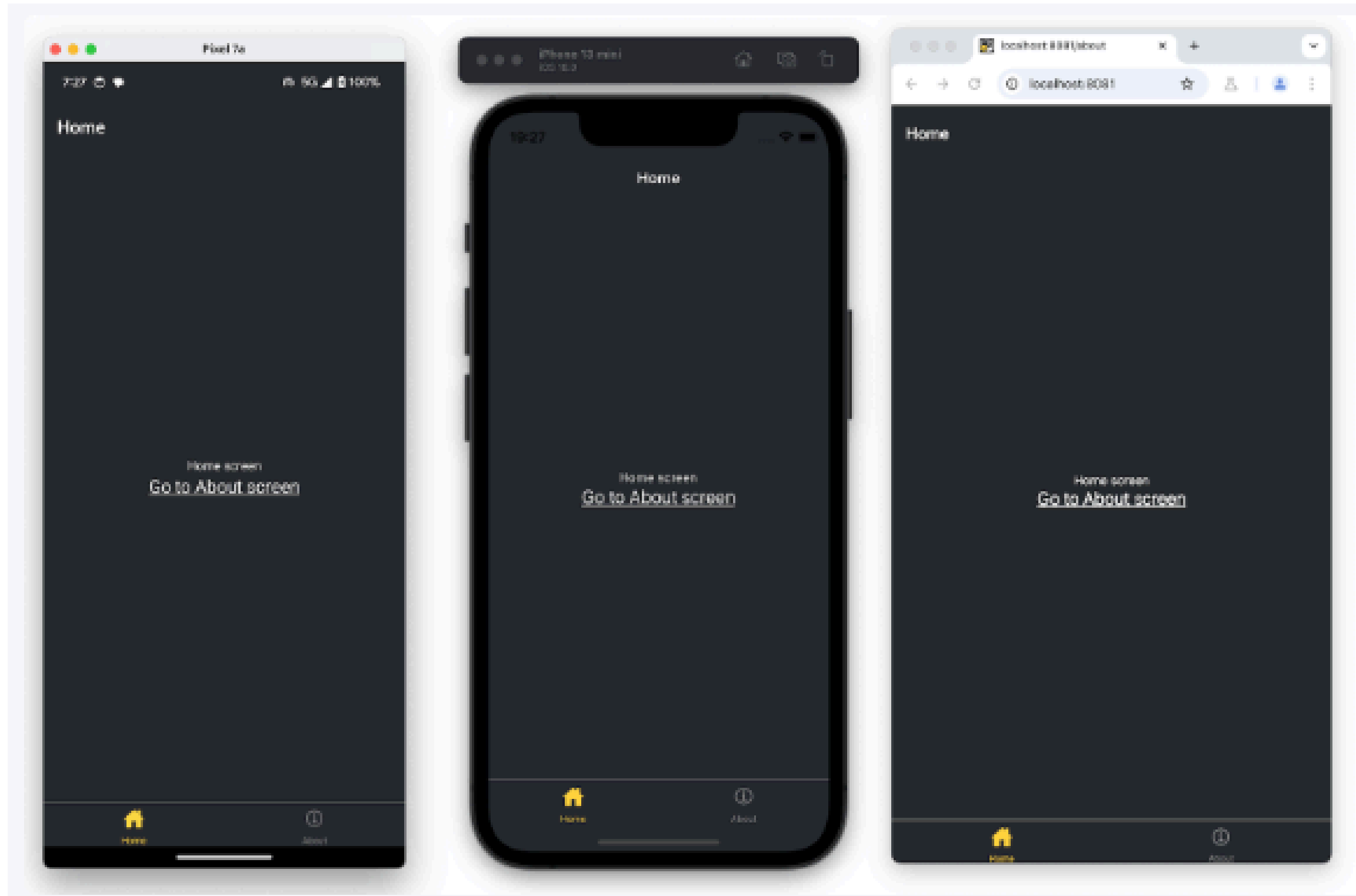
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#25292e',
    justifyContent: 'center',
    alignItems: 'center',
  },
  text: {
    color: '#fff',
  },
});
```

Atualizar a aparência do navegador da guia inferior

Antes:



Depois:



app/(tabs)/_layout.tsx

Copy

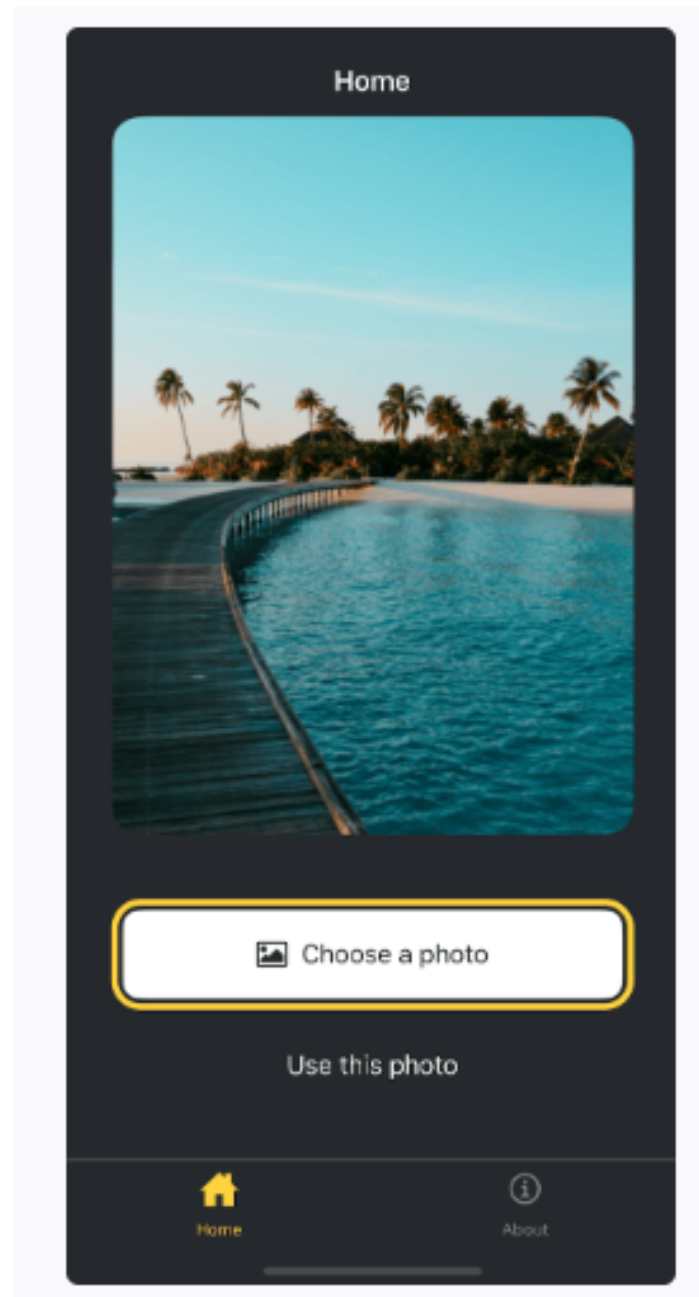
```
import { Tabs } from 'expo-router';
import Ionicons from '@expo/vector-icons/Ionicons';

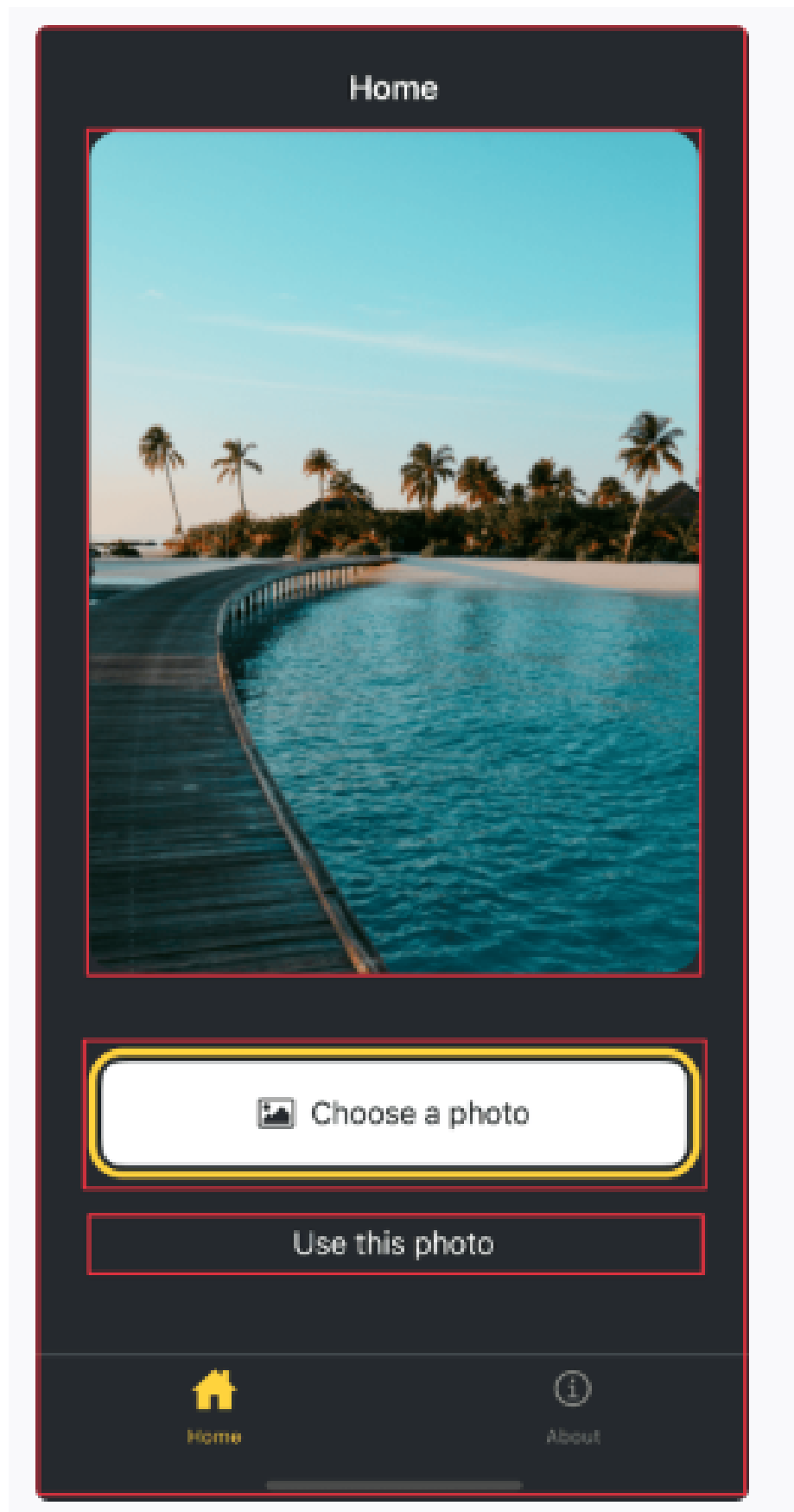
export default function TabLayout() {
  return (
    <Tabs
      screenOptions={{
        tabBarActiveTintColor: '#ffd33d',
      }}
    >
      <Tabs.Screen
        name="index"
        options={{
          title: 'Home',
          tabBarIcon: ({ color, focused }) => (
            <Ionicons name={focused ? 'home-sharp' : 'home-outline'} color={color} size={24} />
          ),
        }}
      />
      <Tabs.Screen
        name="about"
        options={{
          title: 'About',
          tabBarIcon: ({ color, focused }) => (
            <Ionicons name={focused ? 'information-circle' : 'information-circle-outline'} color={color} size={24}/>
          ),
        }}
      />
    </Tabs>
  );
}
```

Atualizar
(tabs)/_layout.tsx

```
<Tabs
  screenOptions={{
    tabBarActiveTintColor: '#ffd33d',
    headerStyle: {
      backgroundColor: '#25292e',
    },
    headerShadowVisible: false,
    headerTintColor: '#fff',
    tabBarStyle: {
      backgroundColor: '#25292e',
    },
  }}
}
```

Construir uma tela

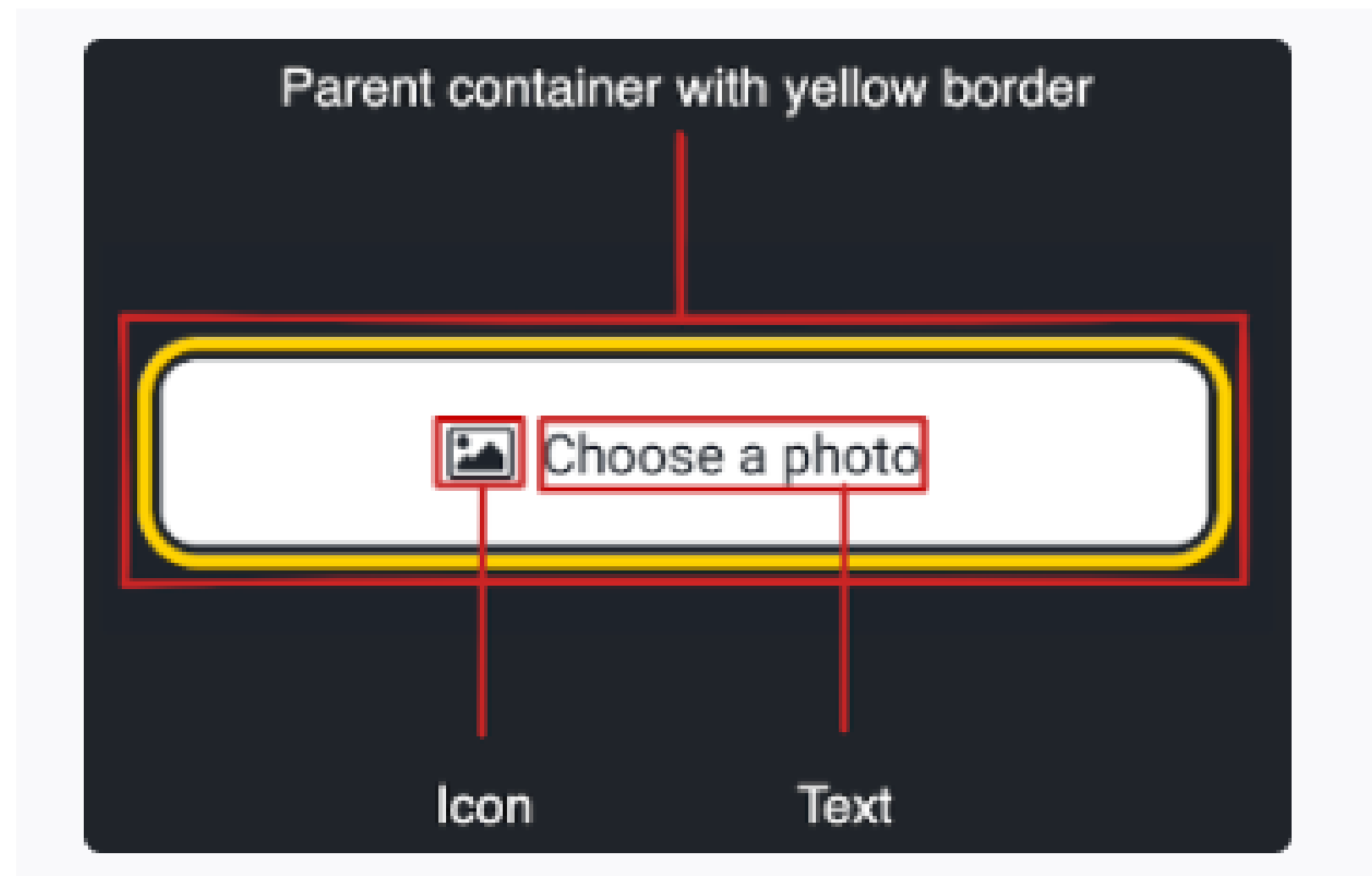




Existem dois elementos essenciais:

- Há uma grande imagem exibida no centro da tela
- Existem dois botões na metade inferior da tela

O primeiro botão contém vários componentes. O elemento pai fornece uma borda amarela e contém um ícone e componentes de texto dentro de uma linha.




Exibir a imagem:

Usaremos expo-imagea biblioteca para exibir a imagem no aplicativo. Ela fornece um componente multiplataforma <Image>para carregar e renderizar uma imagem.

Pare o servidor de desenvolvimento pressionando Ctrl+ cno terminal. Em seguida, instale a expo-imagebiblioteca:

 terminal

 Cópia

```
- npx expo install expo-image
```

Dividir componentes em arquivos:

- Vamos dividir o código em vários arquivos à medida que adicionamos mais componentes a esta tela.

```
components/ImageViewer.tsx

import { ImageSourcePropType, StyleSheet } from 'react-native';
import { Image } from 'expo-image';

type Props = {
  imgSource: ImageSourcePropType;
};

export default function ImageViewer({ imgSource }: Props) {
  return <Image source={imgSource} style={styles.image} />;
}

const styles = StyleSheet.create({
  image: {
    width: 320,
    height: 440,
    borderRadius: 18,
  },
});
```

Atualizar (tabs)/Index

```
app/(tabs)/index.tsx Copy ⋮

import { StyleSheet, View } from 'react-native';

import ImageViewer from '@components/ImageViewer';

const PlaceholderImage = require('@assets/images/background-image.png');

export default function Index() {
  return (
    <View style={styles.container}>
      <View style={styles.imageContainer}>
        <ImageViewer imgSource={PlaceholderImage} />
      </View>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#25292e',
    alignItems: 'center',
  },
  imageContainer: {
    flex: 1,
  },
});
```

O que é a @ declaração no import?

O @ símbolo é um alias de caminho personalizado para importar componentes personalizados e outros módulos, em vez de caminhos relativos. O Expo CLI o configura automaticamente em tsconfig.json .

Crie botões usando Pressable

O React Native inclui alguns componentes diferentes para lidar com eventos de toque, mas `<Pressable>` é recomendado por sua flexibilidade. Ele pode detectar toques únicos, pressionamentos longos, acionar eventos separados quando o botão é pressionado e liberado, e muito mais.

No design, precisamos criar dois botões. Cada um tem um estilo e rótulo diferentes. Vamos começar criando um componente reutilizável para esses botões. Crie um arquivo `Button.tsx` dentro do diretório de componentes com o seguinte código:

 componentes/Button.tsx

 Cópia



```
import { StyleSheet, View, Pressable, Text } from 'react-native';

type Props = {
  label: string;
};

export default function Button({ label }: Props) {
  return (
    <View style={styles.buttonContainer}>
      <Pressable style={styles.button} onPress={() => alert('You pressed a button.')}>
        <Text style={styles.buttonLabel}>{label}</Text>
      </Pressable>
    </View>
  );
}
```

```
const styles = StyleSheet.create({
  buttonContainer: {
    width: 320,
    height: 68,
    marginHorizontal: 20,
    alignItems: 'center',
    justifyContent: 'center',
    padding: 3,
  },
  button: {
    borderRadius: 10,
    width: '100%',
    height: '100%',
    alignItems: 'center',
    justifyContent: 'center',
    flexDirection: 'row',
  },
  buttonLabel: {
    color: '#fff',
    fontSize: 16,
  },
});
```

Atualizar (tabs)/index.tsx

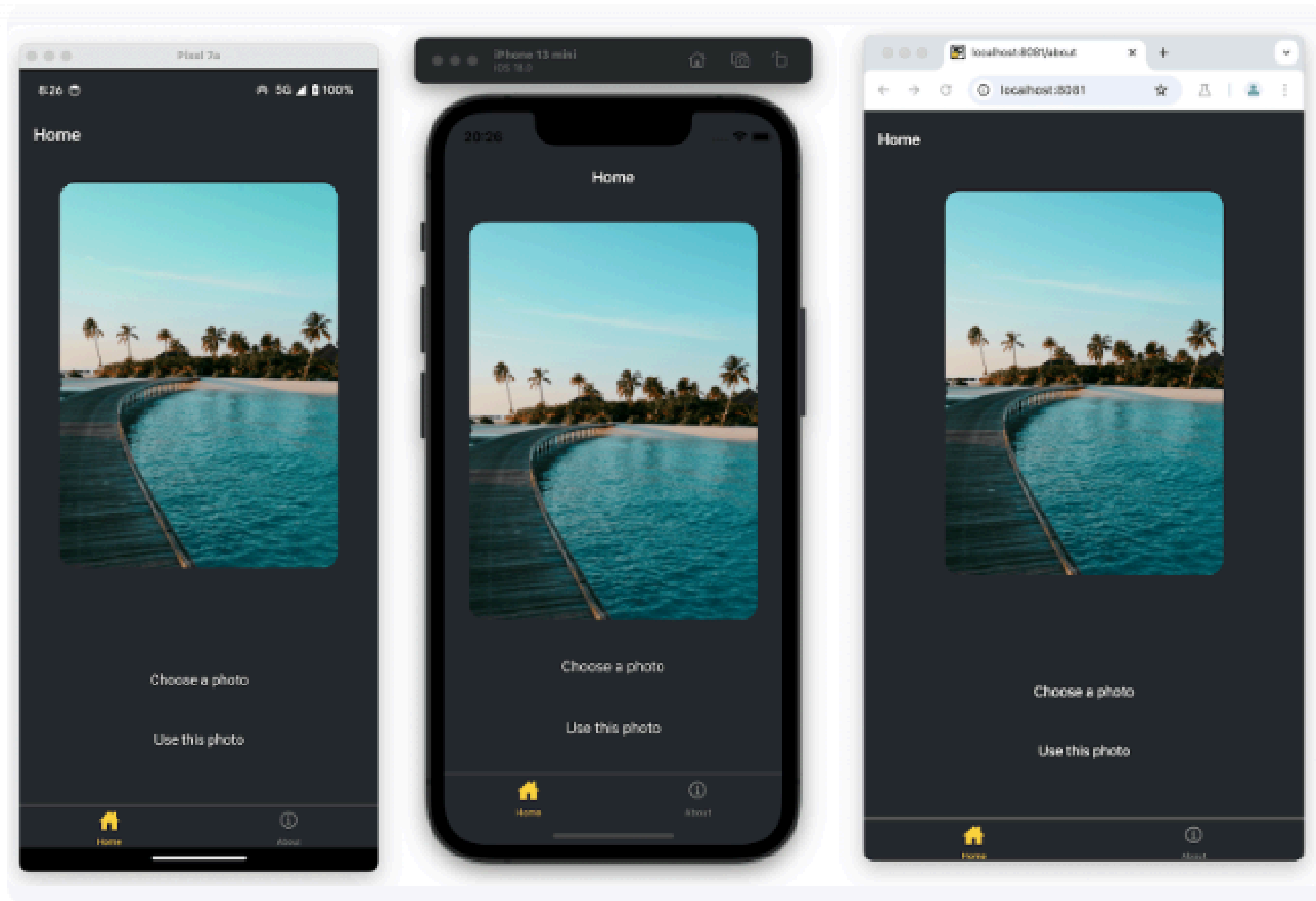
```
import { View, StyleSheet } from 'react-native';

import Button from '@components/Button';
import ImageViewer from '@components/ImageViewer';

const PlaceholderImage = require("@/assets/images/background-image.png");

export default function Index() {
  return (
    <View style={styles.container}>
      <View style={styles.imageContainer}>
        <ImageViewer imgSource={PlaceholderImage} />
      </View>
      <View style={styles.footerContainer}>
        <Button label="Choose a photo" />
        <Button label="Use this photo" />
      </View>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#25292e',
    alignItems: 'center',
  },
  imageContainer: {
    flex: 1,
    paddingTop: 28,
  },
  footerContainer: {
    flex: 1 / 3,
    alignItems: 'center',
  },
});
```

Melhore o componente do botão reutilizável

O botão "Escolher uma foto" requer um estilo diferente do botão "Usar esta foto", então adicionaremos uma nova propriedade de tema de botão que nos permitirá aplicar um `primarytema`. Este botão também tem um ícone antes do rótulo. Usaremos um ícone da `@expo/vector-icons` biblioteca.

Para carregar e exibir o ícone no botão, vamos usar `FontAwesome` biblioteca. Modifique `components/Button.tsx` para adicionar o seguinte trecho de código:

 componentes/Button.tsx

 Cópia



```
import { StyleSheet, View, Pressable, Text } from 'react-native';  
import FontAwesome from '@expo/vector-icons/FontAwesome';  
  
type Props = {  
  label: string;  
  theme?: 'primary';  
};
```

```
export default function Button({ label, theme }: Props) {  
  if (theme === 'primary') {  
    return (  
      <View  
        style={[  
          styles.buttonContainer,  
          { borderWidth: 4, borderColor: '#ffd33d', borderRadius: 18 },  
        ]}>  
        <Pressable  
          style={[styles.button, { backgroundColor: '#fff' }]}  
          onPress={() => alert('You pressed a button.')}>  
          <FontAwesome name="picture-o" size={18} color="#25292e" style={styles.buttonIcon} />  
          <Text style={[styles.buttonLabel, { color: '#25292e' }]}>{label}</Text>  
        </Pressable>  
      </View>  
    );  
  }  
}
```

```
const styles = StyleSheet.create({
  buttonContainer: {
    width: 320,
    height: 68,
    marginHorizontal: 20,
    alignItems: 'center',
    justifyContent: 'center',
    padding: 3,
  },
  button: {
    borderRadius: 10,
    width: '100%',
    height: '100%',
    alignItems: 'center',
    justifyContent: 'center',
    flexDirection: 'row',
  },
  buttonIcon: {
    paddingRight: 8,
  },
  buttonLabel: {
    color: '#fff',
    fontSize: 16,
  },
});
```

- modifique o arquivo app/(tabs)/index.tsx para usar o theme="primary"prop no primeiro botão.

```
export default function Index() {  
  return (  
    <View style={styles.container}>  
      <View style={styles.imageContainer}>  
        <ImageViewer imgSource={PlaceholderImage} />  
      </View>  
      <View style={styles.footerContainer}>  
        <Button theme="primary" label="Choose a photo" />  
        <Button label="Use this photo" />  
      </View>  
    </View>  
  );  
}
```