Lista de Exercícios – Expo / React Native

Professor: André Olímpio

Disciplina: Programação para Dispositivos Móveis I

Exercício 1 – Fazer um aplicativo com um botão que ao ser clicado abre o aplicativo YouTube do dispositivo em um vídeo específico.

Exercício 2 – Fazer um aplicativo com um botão que ao ser clicado abre a interface de discagem para chamadas telefônicas do dispositivo com um número específico.

Exercício 3 – Fazer um aplicativo com um botão que ao ser clicado abre o Instagram da Fatec Jacareí (https://www.instagram.com/fatec_jacarei).

Exercício 4 – Alterar o exemplo de listagem de contatos do dispositivo para listar somente os contatos do telefone cujo nome começa pela letra C.

Dica:

- Faça a filtragem no array disponível na variável `data`:

```
if (data.length > 0) {
  setContacts(data);
}
```

Exercício 5 – Alterar o exemplo de listagem de contatos do dispositivo para listar somente o primeiro nome do contato.

Dica:

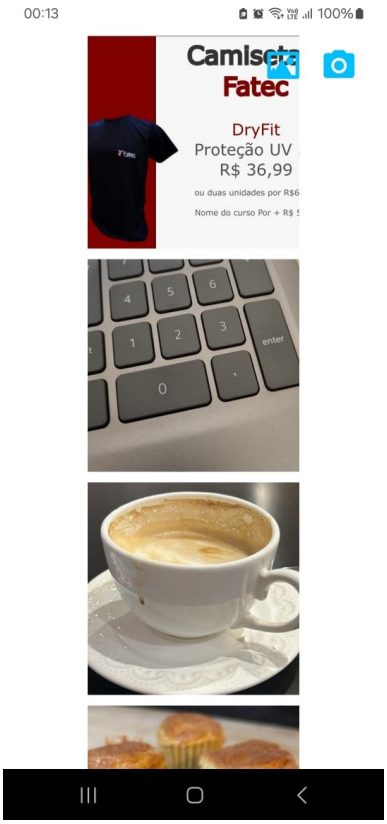
- Use a propriedade `Contacts.Fields.FirstName` no método `Contacts.getContactsAsync`.

Exercício 6 – Alterar os exemplos para ter dois botões no canto superior direito. Um botão abre a galeria de fotos do dispositivo e o outro abre a câmera para tirar uma foto.

Dicas:

- Use um componente `MaterialIcons` com o `name photo` e color `deepskyblue` para criar um botão;
- Use um componente `MaterialIcons` com o `name photo-camera` e color `deepskyblue` para criar um botão;
- Use a altura da `StatusBar` (https://reactnative.dev/docs/statusbar) nos estilos para posicionar os botões no canto superior esquerdo da tela.

Exercício 7 – Alterar o aplicativo do exercício anterior para exibir todas as imagens obtidas pela câmera ou selecionadas na biblioteca do dispositivo.



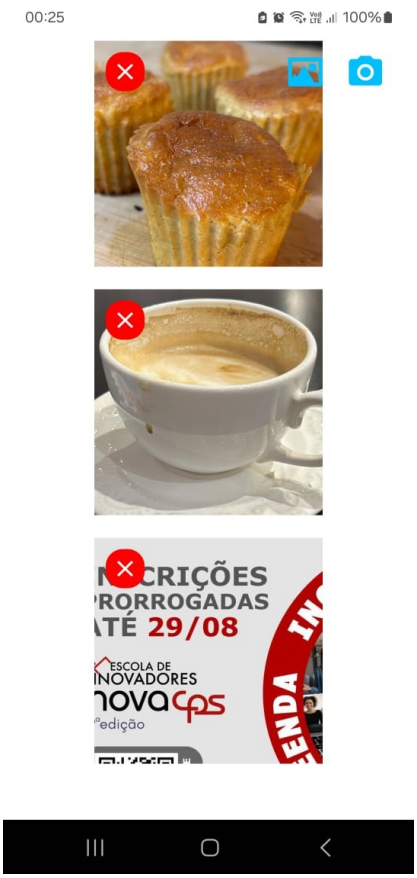
Dicas:

- Crie uma propriedade de estado para manter um array com os URIs das imagens;
- Carregue as imagens em um componente `ScrollView` (https://reactnative.dev/docs/scrollview).

Exercício 8 – Alterar o aplicativo do exercício anterior para incluir um botão no canto superior esquerdo de cada imagem. Ao clicar no botão a imagem será removida.

Dicas:

- Use um componente `MaterialIcons` com o `name close` para criar um botão.



i. Enviar mensagens SMS, WhatsApp e e-mail

No React Native, o módulo Linking é utilizado para interagir com aplicativos e funcionalidades externas ao nosso aplicativo, como abrir URLs, iniciar chamadas telefônicas, enviar mensagens de texto ou e-mails, abrir aplicativos de redes sociais, e outros. Ele fornece uma interface para acessar e manipular links universais ou profundos (deep links). Deep links são URLs que levam o usuário diretamente para um conteúdo específico dentro de um aplicativo.

O uso de Linking permite que o nosso aplicativo inicie o envio de uma mensagem SMS abrindo o aplicativo de mensagens com o número e o conteúdo da mensagem já preenchidos, mas o envio final ainda precisa ser confirmado pelo usuário, ou seja, o código a seguir apenas dispara a abertura do aplicativo de mensagens do dispositivo.

```
import React from 'react';
import { View, Button, Alert, Linking, SafeAreaView } from 'react-native';
import styles from "./styles";

const Sms: React.FC = () => {
  const sendSms = () => {
    const phoneNumber = '1234567890';
    const message = 'Boa noite!\nCorpo da mensagem a ser enviada.\nAtenciosamente.';

    // Criação da URL com o esquema SMS
    const url = `sms:${phoneNumber}?body=${encodeURIComponent(message)}`;
    // Verifica se a URL pode ser aberta (se o dispositivo suporta envio de SMS)
    Linking.canOpenURL(url)
      .then((supported) => {
        if (supported) {
          // Abre o app de SMS padrão com o número e mensagem
          Linking.openURL(url);
        } else {
          Alert.alert('Erro', 'Este dispositivo não suporta envio de SMS.');
        }
      })
      .catch((err) => console.error('Erro ao enviar SMS', err));
  };

  return (
    <SafeAreaView style={styles.container}>
```

```
      <View>
        <Button title="Enviar SMS" onPress={sendSms} />
      </View>
    </SafeAreaView>
  );
};

export default Sms;
```

O Linking cria uma URL que segue o esquema `sms`, que o sistema operacional reconhece e abre o aplicativo de mensagens padrão.

```
`sms:1234567890?body=${encodeURIComponent(message)}`
```

Formato da URL:

- `sms` é o esquema de URL que indica ao sistema que o aplicativo de mensagens deve ser aberto;
- `123456789` é o número de telefone do destinatário;
- `?body=` é o parâmetro que recebe o corpo da mensagem;
- `encodeURIComponent(message)` garante que a mensagem seja codificada corretamente para ser passada como parâmetro na URL.

Nem todos os dispositivos podem suportar o esquema `sms`. Por isso precisamos utilizar o método `canOpenURL` para verificar se o dispositivo pode processar a URL antes de tentar abri-la.

Como o Linking abre outro aplicativo para enviar a mensagem, ele não requer permissões especiais como `SEND_SMS` (https://reactnative.dev/docs/permissionsandroid), que seria necessária para enviar SMS sem a intervenção do usuário.

> O Linking não envia a mensagem automaticamente. Ele apenas abre o aplicativo de mensagens com a mensagem preenchida. O usuário precisa confirmar o envio.

Para enviar mensagens pelo WhatsApp, podemos usar o Linking de maneira semelhante ao envio de SMS. A URL segue o formato `https://wa.me/`, onde `wa.me` é um atalho oficial do WhatsApp para iniciar conversas. O parâmetro `text` permite incluir uma mensagem pré-preenchida.

```
const url = `https://wa.me/${phoneNumber}?text=${encodeURIComponent(message)}`;
```

Código de exemplo para acessar o WhatsApp:

```
import React from 'react';
import { View, Button, Alert, Linking, SafeAreaView } from 'react-native';
import styles from "./styles";
```

```
const Whatsapp: React.FC = () => {
  const sendWhatsapp = () => {
    // Número de telefone no formato internacional (com código do país)
    const phoneNumber = '5512987654321';
    const message = 'Boa noite!\nCorpo da mensagem a ser enviada.\nAtenciosamente.';

    // Criação da URL com o esquema do WhatsApp
    const url = `https://wa.me/${phoneNumber}?text=${encodeURIComponent(message)}`;
    // Verifica se a URL pode ser aberta (se o WhatsApp está instalado)
    Linking.canOpenURL(url)
      .then((supported) => {
        if (supported) {
          /// Abre o WhatsApp com o número e a mensagem preenchidos
          Linking.openURL(url);
        } else {
          Alert.alert('Erro', 'O WhatsApp não está instalado neste dispositivo.');
        }
      })
      .catch((err) => console.error('Erro ao tentar abrir o WhatsApp', err));
  };

  return (
    <SafeAreaView style={styles.container}>
      <View>
        <Button title="Enviar WhatsApp" onPress={sendWhatsapp} />
      </View>
    </SafeAreaView>
  );
};

export default Whatsapp;
```

Para enviar e-mails podemos utilizar o módulo Linking para abrir o cliente de e-mail padrão do dispositivo com o endereço do destinatário, assunto e corpo da mensagem já preenchidos. Assim como é mostrado no código a seguir.

```
import React from 'react';
import { View, Button, Alert, Linking, SafeAreaView } from 'react-native';
import styles from "./styles";

const Mail: React.FC = () => {
  const sendMail = () => {
    // Endereço de e-mail do destinatário
    const to = 'arley.souza@fatec.sp.gov.br';
    const subject = 'Aula de programação'; // Assunto do e-mail
    // Corpo do e-mail
    const body = 'Boa noite.\n\nAula de envio de e-mail.\n\nAtenciosamente.';

    // Criação da URL com o esquema 'mailto'
                                        const                    url                    =
`mailto:${to}?subject=${encodeURIComponent(subject)}&body=${encodeURIComponent(body)}`;

    // Verifica se a URL pode ser aberta (se há um cliente de e-mail configurado)
    Linking.canOpenURL(url)
      .then((supported) => {
        if (supported) {
          // Abre o cliente de e-mail padrão com o destinatário, assunto e corpo preenchidos
          Linking.openURL(url);
        } else {
              Alert.alert('Erro', 'Nenhum aplicativo de e-mail está configurado neste
dispositivo.');
        }
      })
      .catch((err) => console.error('Erro ao tentar abrir o cliente de e-mail', err));
  };

  return (
    <SafeAreaView style={styles.container}>
```

```
      <View>
        <Button title="Enviar e-mail" onPress={sendMail} />
      </View>
    </SafeAreaView>
  );
};

export default Mail;
```

O Linking permite abrir o Google Maps em uma localização específica. Para isso, é necessário utilizar URLs personalizadas que funcionam tanto para o Google Maps quanto para o Apple Maps (em dispositivos iOS).

No exemplo a seguir está a URL para abrir o Google Maps.

```
import React from 'react';
import { View, Button, Alert, Linking, SafeAreaView } from 'react-native';
import styles from "./styles";

const Maps: React.FC = () => {
  const openMap = () => {
    const latitude = -23.295122027241426;
    const longitude = -45.967088557159585;
    const label = "Fatec Jacareí";

    const url = `https://www.google.com/maps/search/?api=1&query=${latitude},${longitude}&query_place_id=${encodeURIComponent(label)}`;

    Linking.canOpenURL(url)
      .then((supported) => {
        if (supported) {
          Linking.openURL(url);
        } else {
          Alert.alert('Erro', 'Não foi possível abrir o Google Maps.');
        }
      })
```

```
      .catch((err) =>
          console.error('Erro ao tentar abrir o aplicativo de mapas', err));
  };

  return (
    <SafeAreaView style={styles.container}>
      <View>
        <Button title="Google Maps" onPress={openMap} />
      </View>
    </SafeAreaView>
  );
};

export default Maps;
```

ii. Acessar a lista de contatos do dispositivo

O pacote expo-contacts (https://www.npmjs.com/package/expo-contacts) é uma biblioteca fornecida pelo Expo que permite acessar os contatos armazenados no dispositivo do usuário em um aplicativo RN.

Principais funcionalidades:

- Solicitar permissões: antes de acessar os contatos, é necessário solicitar permissão ao usuário. O pacote fornece métodos para facilitar essa solicitação;
- Acesso aos contatos: o expo-contacts permite acessar os contatos armazenados no dispositivo, podendo obter detalhes como nome, número de telefone, e-mail, endereço etc.;
- Filtragem e busca: permite buscar contatos específicos ou filtrar a lista de contatos com base em critérios como nome, número de telefone, e-mail etc.

Primeiramente é necessário instalar o pacote:

```
npm i expo-contacts
```

O expo-contacts é compatível com ambas as plataformas (Android e iOS), embora existam algumas diferenças de implementação entre elas.

Operações disponíveis:

- Solicitar permissão para acessar os contatos do dispositivo;

  ```
  await Contacts.requestPermissionsAsync();
  ```

  Se o resultado do pedido de concessão for `granted` (concedido), então podemos acessar a lista de contatos usando o método `getContactsAsync`.

- Ler a lista de contatos armazenados no dispositivo;
- Filtrar contatos por campos específicos, como número de telefone ou e-mail;
- Adicionar, atualizar ou excluir contatos - em alguns casos, dependendo do suporte da plataforma.

O código a seguir é usado para listar todos os contatos do dispositivo:

```
import React, { useState, useEffect } from 'react';
import { Text, View, FlatList, StyleSheet } from 'react-native';
import * as Contacts from 'expo-contacts';
import styles from "./styles";

export default function ContactsComponent() {
  const [contacts, setContacts] = useState([]);
  const [hasPermission, setHasPermission] = useState(null);

  useEffect(() => {
    (async () => {
      const { status } = await Contacts.requestPermissionsAsync();
      setHasPermission(status === 'granted');

      if (status === 'granted') {
        const { data } = await Contacts.getContactsAsync({
          fields: [Contacts.Fields.PhoneNumbers, Contacts.Fields.Emails],
        });

        if (data.length > 0) {
          setContacts(data);
        }
      }
    })();
  }, []);

  if (hasPermission === null) {
    return <View />;
```

```
  }
  if (hasPermission === false) {
    return <Text>No access to contacts</Text>;
  }

  return (
    <View style={styles.container}>
      <FlatList
        data={contacts}
        keyExtractor={(item) => item.id}
        renderItem={({ item }) => (
          <View style={styles.row}>
            <Text style={styles.name}>{item.name}</Text>
            {item.phoneNumbers && item.phoneNumbers.map((phone, index) => (
              <Text key={index} style={styles.number}>{phone.number}</Text>
            ))}
          </View>
        )}
      />
    </View>
  );
}
```

Arquivo: styles.ts

```
import { StyleSheet } from "react-native";
import Constants from "expo-constants";

const styles = StyleSheet.create({
  container: {
    flex: 1,
    paddingTop: Constants.statusBarHeight,
    backgroundColor: "#222",
  },
  scrollViewContent: {
```

```
    paddingHorizontal: 10,
    alignItems: "center",
  },
  row: {
    width: "100%",
    marginTop: 10,
    borderBottomWidth: 1,
    borderBottomColor: "#bbb",
    paddingBottom: 10,
    paddingLeft: 10,
  },
  name: {
    color: "yellow",
  },
  number: {
    color: "#fff",
  },
});

export default styles;
```

iii. Acessar a câmera do dispositivo

O pacote `expo-image-picker` fornece acesso à interface do usuário do sistema para selecionar imagens e vídeos da biblioteca do dispositivo ou tirar uma foto ou gravar um vídeo com a câmera (https://docs.expo.dev/versions/latest/sdk/imagepicker).

O código a seguir permite ao usuário escolher uma imagem da Galeria e exibi-la na tela:

```
import { useState } from "react";
import { Button, Image, View } from "react-native";
import * as ImagePicker from "expo-image-picker";
import styles from "./styles";

export default function Galeria() {
  const [image, setImage] = useState<string | null>(null);
```

```
  const pickImage = async () => {
    // No permissions request is necessary for launching the image library
    let result = await ImagePicker.launchImageLibraryAsync({
      mediaTypes: ImagePicker.MediaTypeOptions.All,
      allowsEditing: true,
      aspect: [4, 3],
      quality: 1,
    });

    console.log(result);

    if (!result.canceled) {
      setImage(result.assets[0].uri);
    }
  };

  return (
    <View style={styles.container}>
      <View style={styles.buttonContainer}>
        <Button title="Escolha uma imagem" onPress={pickImage} />
        {image && <Image source={{ uri: image }} style={styles.image} />}
      </View>
    </View>
  );
}
```

Arquivo de estilos styles.ts:

```
import { StyleSheet } from "react-native";
import Constants from "expo-constants";

const styles = StyleSheet.create({
```

```
  container: {
    flex: 1,
    paddingTop: Constants.statusBarHeight,
    backgroundColor: "#222",
    justifyContent: "center"
  },
  buttonContainer: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    marginBottom: 20,
  },
  image: {
    width: 300,
    height: 300,
    marginTop: 20,
  },
});


export default styles;
```

O código anterior é utilizado para o usuário selecionar uma imagem da Galeria. O código seguir será usado para ativar a câmera do dispositivo para tirar uma foto. O primeiro passo é substituir o método `launchImageLibraryAsync` pelo método `launchCameraAsync` do `expo-image-picker`. Além disso, é necessário solicitar permissões de câmera antes de usá-la.

O método `ImagePicker.requestCameraPermissionsAsync()` é utilizado para pedir ao usuário a permissão de uso da câmera. O status da permissão é armazenado no estado `hasCameraPermission`.

```
import { useState, useEffect } from "react";
import { Button, Image, View, Alert } from "react-native";
import * as ImagePicker from "expo-image-picker";
import styles from "./styles";


export default function Photo() {
  const [image, setImage] = useState<string | null>(null);
```

```
const [hasCameraPermission, setHasCameraPermission] = useState<boolean | null>(null);

useEffect(() => {
  (async () => {
    const { status } = await ImagePicker.requestCameraPermissionsAsync();
    setHasCameraPermission(status === "granted");
  })();
}, []);

const takePhoto = async () => {
  if (hasCameraPermission === null) {
    return;
  }

  if (hasCameraPermission === false) {
    Alert.alert("Sem permissão para acessar a câmera");
    return;
  }

  let result = await ImagePicker.launchCameraAsync({
    allowsEditing: true,
    aspect: [4, 3],
    quality: 1,
  });

  console.log(result);

  if (!result.canceled) {
    setImage(result.assets[0].uri);
  }
};

return (
```

```
    <View style={styles.container}>
      <View style={styles.buttonContainer}>
        <Button title="Tirar uma foto" onPress={takePhoto} />
        {image && <Image source={{ uri: image }} style={styles.image} />}
      </View>
    </View>
  );
}
```

O código anterior é usado para ativar a câmera do dispositivo para tirar uma foto. O código a seguir é usado para ativar a câmera para gravar vídeos com som. Para alterar o código para gravar um vídeo em vez de tirar uma foto, o método `launchCameraAsync` recebe a propriedade `mediaTypes` configurada para `ImagePicker.MediaTypeOptions.Videos`:

Para foto:

```
result = await ImagePicker.launchCameraAsync({
  allowsEditing: true,
  aspect: [4, 3],
  quality: 1,
});
```

Para vídeo:

```
result = await ImagePicker.launchCameraAsync({
  mediaTypes: ImagePicker.MediaTypeOptions.Videos,
  allowsEditing: true,
  quality: 1,
});
```

Para gravar vídeos, além das permissões de câmera, é necessário solicitar permissões de microfone. Para ambos será utilizado o pacote `expo-av` (https://www.npmjs.com/package/expo-av).

```
import { useState, useEffect } from "react";
import { Button, View, Alert } from "react-native";
import * as ImagePicker from "expo-image-picker";
import { Video } from "expo-av";
import styles from "./styles";

export default function Camera() {
  const [video, setVideo] = useState<string | null>(null);
  const [hasCameraPermission, setHasCameraPermission] = useState<
    boolean | null
  >(null);
```

```
useEffect(() => {
  (async () => {
    const { status } = await ImagePicker.requestCameraPermissionsAsync();
    setHasCameraPermission(status === "granted");
  })();
}, []);

const recordVideo = async () => {
  if (hasCameraPermission === null) {
    return;
  }

  if (hasCameraPermission === false) {
    Alert.alert("Sem permissão para acessar a câmera");
    return;
  }

  let result = await ImagePicker.launchCameraAsync({
    mediaTypes: ImagePicker.MediaTypeOptions.Videos,
    allowsEditing: true,
    quality: 1,
  });

  console.log(result);

  if (!result.canceled) {
    setVideo(result.assets[0].uri);
  }
};

return (
  <View style={styles.container}>
```

```
      <View style={styles.buttonContainer}>
        <Button title="Gravar um vídeo" onPress={recordVideo} />
        {video && (
          <Video
            source={{ uri: video }}
            style={styles.video}
            useNativeControls
            resizeMode="contain"
            isLooping
          />
        )}
      </View>
    </View>
  );
}
```