Lecture 11

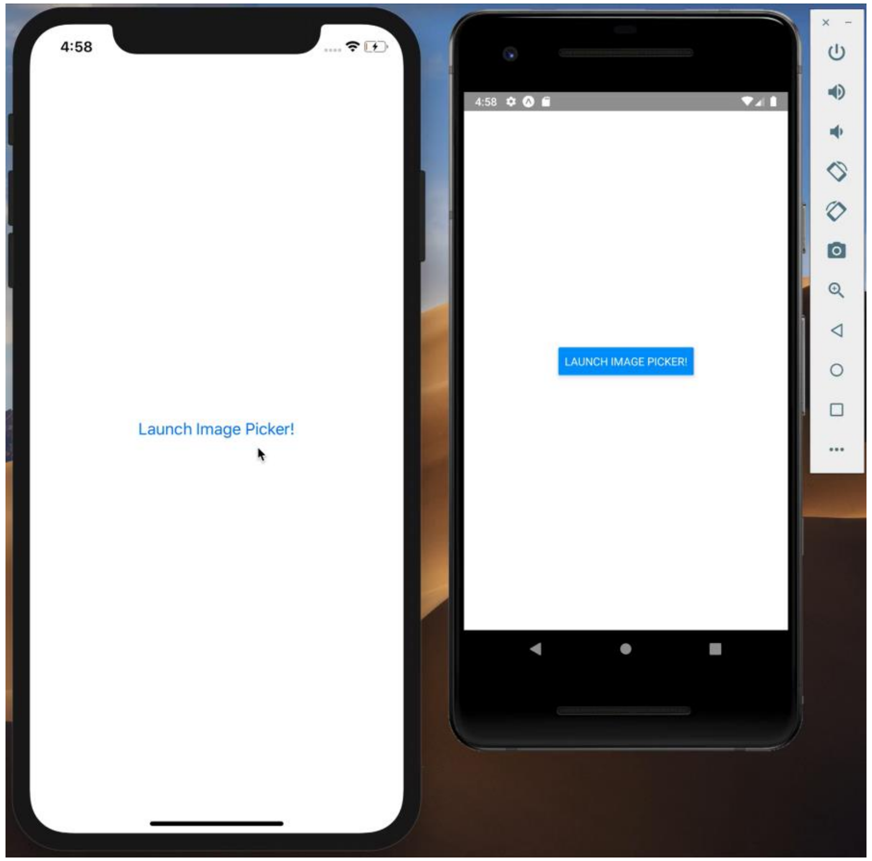
React Native Media

Content

- Expo ImagePicker
- Expo ImageManipulator
- Expo Video
- Expo Camera

ImagePicker

- A library that provides access to the system's UI for selecting images and videos from the phone's library or taking a photo with the camera.



expo-image-picker

- `expo-image-picker` provides access to the system's UI for selecting images and videos from the phone's library or taking a photo with the camera.
- Installation

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

- **On iOS**, when an image (usually of a higher resolution) is picked from the camera roll, the result of the cropped image gives the wrong value for the cropped rectangle in some cases.Unfortunately, this issue is with the underlying `UIImagePickerController` due to a bug in the closed-source tools built into iOS.

`app.json` with plugin config

```
 "expo": {
      "plugins": [
        [
          "expo-image-picker",
          {
            "photosPermission": "The app accesses your photos to let you share them with your friends."
          }
        ]
      ]
    }
```

Configuration in app config

- This configuration in the `app.json` file is used when using the Expo Image Picker
- The entry "`expo-image-picker`" indicates that you are using the Expo Image Picker library, and you are configuring it through the Expo plugin system.
- The "`photosPermission`" field specifies the message that will be shown to the user when the app requests access to their photos.
- The value "`The app accesses your photos to let you share them with your friends.`" is a description that appears when the permission prompt is shown.
- This message helps users understand why the app needs access to their photos, improving transparency and user trust.

Configurable properties

- **`photosPermission`** :
  - Default: "`Allow $(PRODUCT_NAME) to access your photos`"
  - Only for IOS : A string to set the NSPhotoLibraryUsageDescription permission message.
- **`cameraPermission`**
  - Default: "`Allow $(PRODUCT_NAME) to access your camera`"
  - Only for IOS: A string to set the NSCameraUsageDescription permission message.
- **`microphonePermission`**
  - Default: "`Allow $(PRODUCT_NAME) to access your microphone`"
  - Only for IOS: A string to set the NSMicrophoneUsageDescription permission message.

Using Image Picker

- Import the `expo-image-picker` library to handle image selection:

```
import * as ImagePicker from 'expo-image-picker';
```

- Use the `useState` hook to manage the state of the selected image:

```
const [image, setImage] = useState(null);
```

- Define an asynchronous function to open the phone’s image library:

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

Checking Permissions for iOS

```
import { askAsync, MEDIA_LIBRARY } from 'expo-permissions';

const pickImage = async () => {
  // Request permission to access the photo library
  const { status } = await askAsync(MEDIA_LIBRARY);
  if (status !== 'granted') {
    alert('Sorry, we need camera roll permissions to make this work!');
    return;
  }

  // Launch the image library
  let result = await ImagePicker.launchImageLibraryAsync({
    mediaTypes: ['images', 'videos'], allowsEditing: true, aspect: [4, 3], quality: 1
  });

  if (!result.cancelled) { console.log(result.assets[0].uri); }
}
```

Using Image Picker

- The function `pickImage` opens the device's gallery, allowing users to pick images or videos.
- If the user selects a file, the function checks if the operation was not canceled

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

- The selected image's URI is stored in the state.

Using Image Picker

- A button is displayed to trigger the image picker
- If an image is selected, it will be displayed in an Image component:

```
<View style={styles.container}>
  <Button title="Pick an image from camera roll" onPress={pickImage} />
  {image && <Image source={{ uri: image }} style={styles.image} />}
</View>
```

Using Image Picker

- When you run the `pickImage` function and pick an image, you will see the image that you picked show up in your app, and a similar log will be shown in the console:

```
  const pickImage = async () => {
    // No permissions request is necessary for
launching the image library
    let result = await
ImagePicker.launchImageLibraryAsync({
      mediaTypes: ['images', 'videos'],
      allowsEditing: true,
      aspect: [4, 3],
      quality: 1,
    });
    console.log(result);
    if (!result.canceled) {
      setImage(result.assets[0].uri);
    }
  };
```

```
{
   "assets": [
     {
       "assetId": "C166F9F5-B5FE-4501-9531",
       "base64": null,
       "duration": null,
       "exif": null,
       "fileName": "IMG.HEIC",
       "fileSize": 6018901,
       "height": 3025,
       "type": "image",
       "uri":
"file:///data/user/0/host.exp.exponent/cache/cropped1814158652.j
pg"
       "width": 3024
     }
   ],
   "canceled": false
 }
```

Expo `ImageManipulator`

- The `ImageManipulator` is a library in the Expo ecosystem that allows you to manipulate images directly in your React Native app.
- It provides a set of functions to perform various image operations, such as rotating, flipping, cropping, and resizing images.
- Installation

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

Key Features

- **Image Transformation**
  - Rotate images to a specific angle.
  - Flip images horizontally or vertically.
  - Crop images to a specified region.
  - Resize images to new dimensions.
- **Image Format**
  - Save images as JPEG or PNG.
  - Adjust the compression quality.
- **Integration**
  - Works well with images captured from the camera or selected from the gallery.
  - Can be used with other Expo libraries like `expo-image-picker`.

Using `ImageManipulator`

- This will first rotate the image 90 degrees clockwise, then flip the rotated image vertically and save it as a PNG.
- Import Statements:

```
import { useState } from 'react';
import { Button, Image, StyleSheet, View } from 'react-native';
import { Asset } from 'expo-asset';
import { FlipType, SaveFormat, useImageManipulator } from 'expo-
image-manipulator';
```

- `Asset` from expo-asset: Manages image assets.
- `FlipType`, `SaveFormat`, `useImageManipulator` from **expo-image-manipulator**:
  - `FlipType`: Enum for flipping direction (Horizontal or Vertical).
  - `SaveFormat`: Enum for saving image format (PNG, JPEG).
  - `useImageManipulator`: Custom hook to manipulate images.

Using `ImageManipulator`

- Loading the Image

```
const IMAGE = Asset.fromModule(require('./assets/avatar.jpg'));
```

- Loads the image `avatar.jpg` from the assets folder.
- Uses `Asset.fromModule` to convert the image into an asset object.
- The image object will have properties like `uri`, which is the local path to the image.

Using `ImageManipulator`

```
export default function App() {
      const [image, setImage] = useState(IMAGE);
```

Uses `useState` to store the current image.

- Initially, it sets the image to the **loaded avatar**.

- `useImageManipulator` creates a manipulation context for the image.
- This function is triggered when the **"Rotate and Flip"** button is pressed.

```
const context = useImageManipulator(IMAGE.uri);
  const rotate90andFlip = async () => {

   context.rotate(90).flip(FlipType.Vertical);

   const image = await context.renderAsync();

   const result = await image.saveAsync({

     format: SaveFormat.PNG,});
    setImage(result);};}
```

Using `ImageManipulator`

- Uses the manipulated image URI (`image.localUri` or `image.uri`).
- Uses the Image component to display the photo.
- Button: Calls `rotate90andFlip` when pressed.

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

    <View style={styles.imageContainer}>

      <Image source={{ uri: image.localUri || image.uri }} style={styles.image} />

    </View>

    <Button title="Rotate and Flip" onPress={rotate90andFlip} />

</View>
```

Expo Video

- The `Expo Video` component is part of the `expo-av` library, which is used to handle audio and video playback in React Native applications.
- It provides a comprehensive set of features for playing videos and controlling playback within Expo-managed apps.
- **Installation**

```
npx expo install expo-video
```

Configuration in app config

- Example `app.json` with config plugin

```json
{
  "expo": {
    "plugins": [
      [
        "expo-video",
        {
          "supportsBackgroundPlayback": true,
          "supportsPictureInPicture": true
        }
      ]
    ],
  }
}
```

Configuration in app config

`supportsBackgroundPlayback`: < Only for: iOS >

- A boolean value to enable background playback on iOS.
- If `true`, the `audio` key is added to the `UIBackgroundModes` array in the `Info.plist` file.
- If `false`, the key is removed. When `undefined`, the key is not modified.

- `supportsPictureInPicture`
  - A boolean value to enable Picture-in-Picture on Android and iOS.
  - If `true`, enables the `android:supportsPictureInPicture` property on Android and adds the audio key to the `UIBackgroundModes` array in the Info.plist file on iOS.
  - If `false`, the key is removed. When `undefined`, the configuration is not modified.

Playing local media from the assets directory

- `expo-video` supports playing local media loaded using the `require` function. You can use the result as a source directly, or assign it to the `assetId` parameter of a `VideoSource` if you also want to configure other properties.

```
import { VideoSource } from 'expo-video';

const assetId = require('./assets/bigbuckbunny.mp4');

const videoSource: VideoSource = {

  assetId,

  metadata: {

    title: 'Big Buck Bunny',

    artist: 'The Open Movie Project'}};

const player1 = useVideoPlayer(assetId); // You can use the `asset` directly as a
video source

const player2 = useVideoPlayer(videoSource);
```

Preloading videos

- While another video is playing, a video can be loaded before showing it in the view. This allows for quicker transitions between subsequent videos and a better user experience.
- To preload a video, you have to create a `VideoPlayer` with a video source. Even when the player is not connected to a `VideoView`, it will fill the buffers. Once it is connected to the `VideoView`, it will be able to start playing without buffering.
- In some cases, it is beneficial to preload a video later in the screen lifecycle. In that case, a `VideoPlayer` with a `null` source should be created. To start preloading, replace the player source with a video source using the `replace()` function.

Preloading videos

- `useVideoPlayer`: A hook to manage video playback.
- `VideoView`: A component to render the video player.
- `VideoSource`: Represents the video source (URL or local file).

```
    import { useVideoPlayer, VideoView, VideoSource }
from 'expo-video';
```

- Uses VideoSource to specify the type. These videos will be used later for playback.

```
  const bigBuckBunnySource: VideoSource = '...';
  const elephantsDreamSource: VideoSource = '...';
```

Preloading videos

- Uses `useVideoPlayer` to initialize video players.

```
export default function PreloadingVideoPlayerScreen() {
  const player1 = useVideoPlayer(bigBuckBunnySource, player => {
    player.play();
  });
  const player2 = useVideoPlayer(elephantsDreamSource, player => {
    player.currentTime = 20;
  });
}
```

Preloading videos

- Switching Between Players

```
export default function PreloadingVideoPlayerScreen() {
  //Switching Between Players
  const [currentPlayer, setCurrentPlayer] = useState(player1);
    const replacePlayer = useCallback(async () => { currentPlayer.pause();
    if (currentPlayer === player1) {
      setCurrentPlayer(player2);
      player1.pause();
      player2.play();
    } else {
      setCurrentPlayer(player1);
      player2.pause();
      player1.play();
    }}, [player1, currentPlayer]);}
```

Preloading videos

- Renders the video player using `VideoView`.
  - `player={currentPlayer}` binds the active player.
  - `nativeControls={false}` disables default controls.
- A button to **replace the current player**:
  - On press, calls `replacePlayer()` to toggle between videos.

```
return (
  <View style={styles.contentContainer}>
    <VideoView player={currentPlayer} style={styles.video} nativeControls={false} />
    <TouchableOpacity style={styles.button} onPress={replacePlayer}>
      <Text style={styles.buttonText}>Replace Player</Text>
    </TouchableOpacity>
  </View>
);
```

Using the VideoPlayer directly

- In most cases, the `useVideoPlayer` hook should be used to create a `VideoPlayer` instance.
- It manages the player's lifecycle and ensures that it is properly disposed of when the component is unmounted. However, in some advanced use cases, it might be necessary to create a `VideoPlayer` that does not get automatically destroyed when the component is unmounted.
- In those cases, the `VideoPlayer` can be created using the `createVideoPlayer` function.
- You need be aware of the risks that come with this approach, as it is your responsibility to call the `release()` method when the player is no longer needed. If not handled properly, this approach may lead to memory leaks.

```
import { createVideoPlayer } from 'expo-video';
const player = createVideoPlayer(videoSource);
```

Receiving events

- The changes in properties of the `VideoPlayer` do not update the React state. Therefore, to display the information about the current state of the `VideoPlayer`, it is necessary to listen to the `events` it emits.
- The event system is based on the `EventEmitter` class and `hooks` from the `expo` package. There are a few ways to listen to events:
- `useEvent` hook: Creates a listener that will return a stateful value that can be used in a component. It also cleans up automatically when the component unmounts.

```
 import { useEvent } from 'expo';

 // ... Other imports, definition of the component, creating the
player etc.

 const { status, error } = useEvent(player, 'statusChange', {
status: player.status });

 // Rest of the component...
```

Receiving events

- `useEventListener` hook
- Built around the `Player.addListener` and `Player.removeListener` methods, creates an event listener with automatic cleanup.

```
import { useEventListener } from 'expo';
// ...Other imports, definition of the component, creating the player etc.
useEventListener(player, 'statusChange', ({ status, error }) => {
  setPlayerStatus(status);
  setPlayerError(error);
  console.log('Player status changed: ', status);
});
// Rest of the component...
```

Receiving events

- `Player.addListener` method
- Most flexible way to listen to events, but requires manual cleanup and more boilerplate code.

```
// ...Imports, definition of the component, creating the player etc.
useEffect(() => {
  const subscription = player.addListener('statusChange', ({ status, error }) => {
    setPlayerStatus(status);
    setPlayerError(error);
    console.log('Player status changed: ', status);
  });
    return () => {
    subscription.remove();
  };}, []);
// Rest of the component...
```

Expo Camera

- `expo-camera` provides a React component that renders a preview of the device's front or back camera.
- The camera's parameters such as zoom, torch, and flash mode are adjustable.
- Using `CameraView`, you can take photos and record videos that are saved to the app's cache.
- The component is also capable of detecting bar codes appearing in the preview.

Expo Camera

- Installation
  - `npx expo install expo-camera`
- Configuration in app config
- The plugin allows you to configure various properties that cannot be set at runtime and require building a new app binary to take effect.

```
    "expo": {
      "plugins": [
        ["expo-camera",{
            "cameraPermission": "Allow $(PRODUCT_NAME) to access
your camera",
            "microphonePermission": "Allow $(PRODUCT_NAME) to
access your microphone",
            "recordAudioAndroid": true}]]}
```

Permissions

- **Android**
  - This package automatically adds the CAMERA permission to your app. If you want to record videos with audio, you have to include the `RECORD_AUDIO` in your app.json inside the expo.android.permissions array.
  - `CAMERA`: Required to be able to access the camera device.
  - `RECORD_AUDIO`: Allows an application to record audio.

```
"android": {
        "permissions": [
          "CAMERA",
          "RECORD_AUDIO"
        ]
  },
```

Permissions

- **iOS**
  - To configure **Camera** and **Microphone** permissions on iOS in your Expo project, you need to declare them in the **app.json** file as follows:
  - **`NSCameraUsageDescription`**: A message that tells the user why the app is requesting access to the device's camera.
  - **`NSMicrophoneUsageDescription`**: A message that tells the user why the app is requesting access to the device's microphone.

Configurable properties

- **`cameraPermission`** :
  - Default: "`Allow $(PRODUCT_NAME) to access your camera`"
  - Only for IOS : A string to set the NSCameraUsageDescription permission message.
- **`microphonePermission`**
  - Default: "`Allow $(PRODUCT_NAME) to access your camera`"
  - Only for IOS: A string to set the NSCameraUsageDescription permission message.
- **`recordAudioAndroid`**
  - Default: "`Allow $(PRODUCT_NAME) to access your microphone`"
  - Only for IOS: A string to set the NSMicrophoneUsageDescription permission message.

Using Expo Camera

- App use Expo Camera allows the user to:
  - Open/close the camera.
  - Toggle between front and back cameras.
  - Request camera permissions.
- Import Statements

```
import { CameraView, CameraType, useCameraPermissions
} from 'expo-camera';
```

  - `useCameraPermissions` is used to request camera access.

Using Expo Camera

- State Initialization

```
const [facing, setFacing] = useState('back');
const [cameraActive, setCameraActive] = useState(false);
const [permission, requestPermission] = useCameraPermissions();
```

- `facing`: Determines whether the camera is front or back.
- `cameraActive`: Tracks whether the camera is open or closed.
- `permission`: Stores the camera permission status.
- `requestPermission`: Function to request camera access.

Using Expo Camera

- Permission Handling

```
if (!permission) {
  // Camera permissions are still loading.
  return <View />;
}
if (!permission.granted) {
  // Camera permissions are not granted yet.
  return (
    <View style={styles.container}>
      <Text style={styles.message}>We need your permission to show the camera</Text>
      <Button onPress={requestPermission} title="Grant Permission" />
    </View>
  );
}
```

Using Expo Camera

- Function to Flip the Camera

```
 function toggleCameraFacing() {
setFacing((current) => (current === 'back' ? 'front' : 'back'));}
```

  - Switches between the **front** and **back** cameras.

- Function to Toggle Camera Visibility

```
function toggleCamera() {
  setCameraActive((current) => !current);
}
```

Using Expo Camera

- Conditional Rendering - Camera View:

```
{cameraActive && (
    <CameraView style={styles.camera} facing={facing}>
            ......
   </CameraView>
 )}
```

- Uses a **conditional statement** to check if the camera is active:
  - If `cameraActive` is **true**, it renders the camera view.
- `<CameraView>` is a custom camera component from the `expo-camera` library.
- `style = {styles.camera}`: Applies a style that takes up the entire screen.
- `facing = {facing}`: Determines which camera (front or back) is used.
  - The value of `facing` can be either **'front'** or **'back'**.

Using Expo Camera

- Flip Camera Button

```
<TouchableOpacity style={styles.button} onPress={toggleCamera}>
   <Text style={styles.text}>Close Camera</Text>
</TouchableOpacity>
```

- Uses **TouchableOpacity** for a **clickable button** with visual feedback.
- Inside `onPress={toggleCameraFacing}`: Calls the function to switch between front and back cameras.
- , there is a Text component displaying "Flip Camera".
- Uses the button and text styles to make the button centered and readable.

Using Expo Camera

- Open Camera Button (If Camera is Inactive):

```
{!cameraActive && (
      <Button title="Open Camera" onPress={toggleCamera} />
)}
```

- This button appears only when the camera is not active.
- Uses the standard React Native Button component.
  - `title="Open Camera"`: Displays the button label.
  - `onPress={toggleCamera}`: Opens the camera by toggling the state.