

**UNIVERSITY OF ENGINEERING & TECHNOLOGY,
TAXILA**



DEPARTMENT OF SOFTWARE ENGINEERING

Mobile Application Development (Quiz 02)

SUBMITTED BY

Hina Azhar (21-SE-51)

SUBMITTED TO

Dr. Kanwal Yousaf

COURSE

MAD

SECTION

Alpha

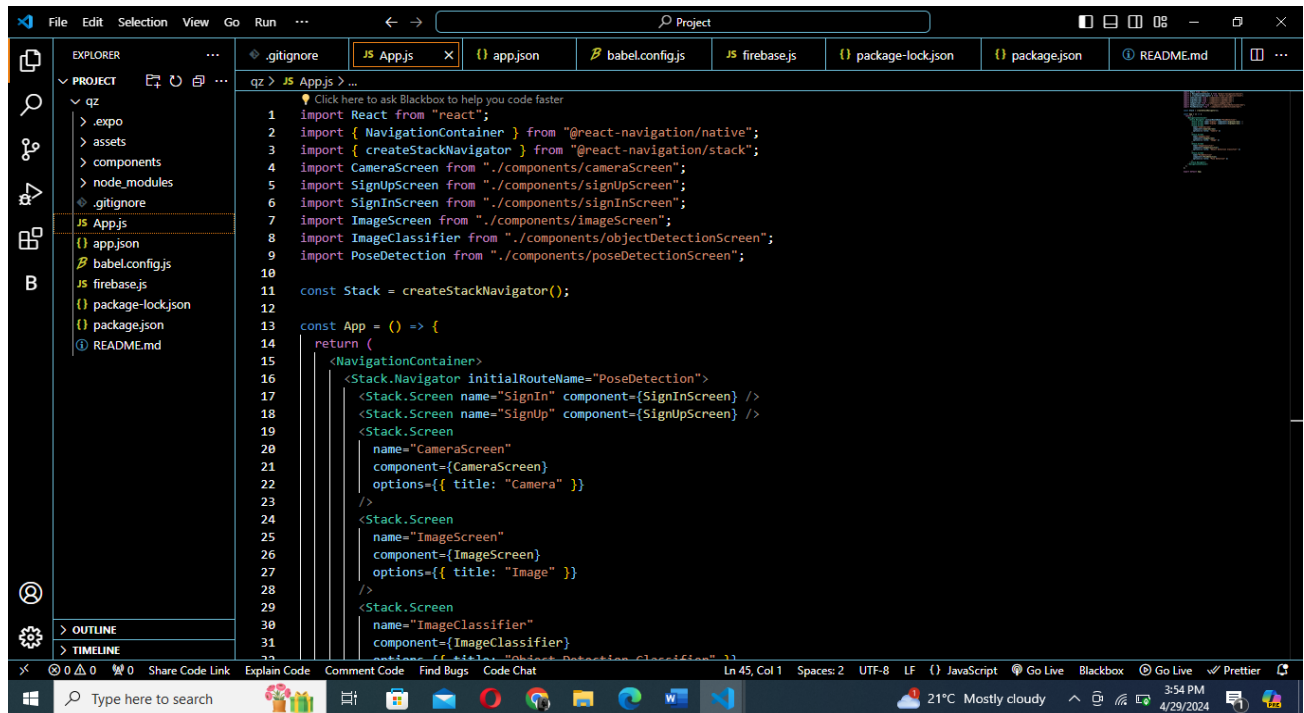
Question # 1

Solution :

Github link:

<https://github.com/HinaAzhar/MAD-Quiz2.git>

App.js:



```
1  import React from "react";
2  import { NavigationContainer } from "@react-navigation/native";
3  import { createStackNavigator } from "react-navigation-stack";
4  import CameraScreen from "../components/cameraScreen";
5  import SignUpScreen from "../components/signUpScreen";
6  import SignInScreen from "../components/signInScreen";
7  import ImageScreen from "../components/imageScreen";
8  import ImageClassifier from "../components/objectDetectionScreen";
9  import PoseDetection from "../components/poseDetectionScreen";
10
11  const Stack = createStackNavigator();
12
13  const App = () => {
14    return (
15      <NavigationContainer>
16        <Stack.Navigator initialRouteName="PoseDetection">
17          <Stack.Screen name="SignIn" component={SignInScreen} />
18          <Stack.Screen name="SignUp" component={SignUpScreen} />
19          <Stack.Screen
20            name="CameraScreen"
21            component={CameraScreen}
22            options={{ title: "Camera" }}
23          />
24          <Stack.Screen
25            name="ImageScreen"
26            component={ImageScreen}
27            options={{ title: "Image" }}
28          />
29          <Stack.Screen
30            name="ImageClassifier"
31            component={ImageClassifier}
32            options={{ title: "Object Detection Classifier" }}
33          />
34          <Stack.Screen
35            name="PoseDetection"
36            component={PoseDetection}
37            options={{ title: "Pose Detection" }}
38          />
39        </Stack.Navigator>
40      </NavigationContainer>
41    );
42  };
43
44  export default App;
```



```
13  const App = () => {
14    return (
15      <NavigationContainer>
16        <Stack.Navigator initialRouteName="PoseDetection">
17          <Stack.Screen name="SignIn" component={SignInScreen} />
18          <Stack.Screen name="SignUp" component={SignUpScreen} />
19          <Stack.Screen
20            name="CameraScreen"
21            component={CameraScreen}
22            options={{ title: "Camera" }}
23          />
24          <Stack.Screen
25            name="ImageScreen"
26            component={ImageScreen}
27            options={{ title: "Image" }}
28          />
29          <Stack.Screen
30            name="ImageClassifier"
31            component={ImageClassifier}
32            options={{ title: "Object Detection Classifier" }}
33          />
34          <Stack.Screen
35            name="PoseDetection"
36            component={PoseDetection}
37            options={{ title: "Pose Detection" }}
38          />
39        </Stack.Navigator>
40      </NavigationContainer>
41    );
42  };
43
44  export default App;
```

`<Stack.Screen`

`name="PoseDetection"`

`component={PoseDetection}`

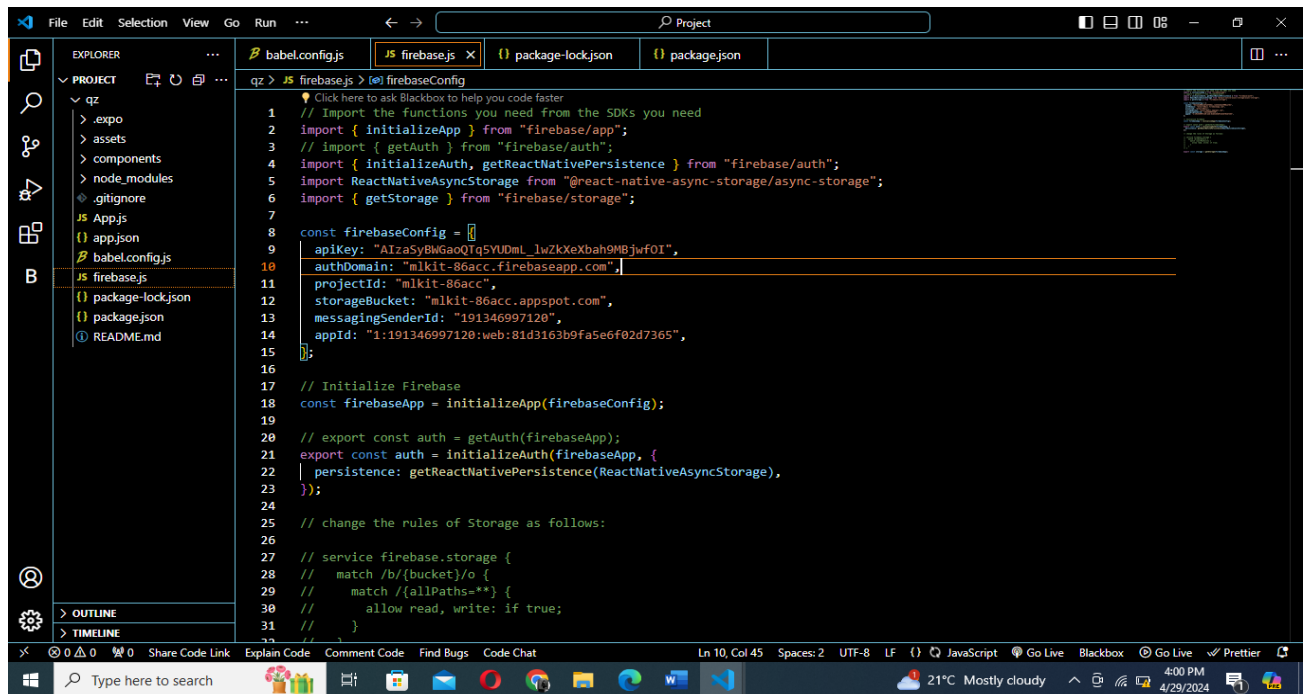
`options={{ title: "Pose Detection" }}`

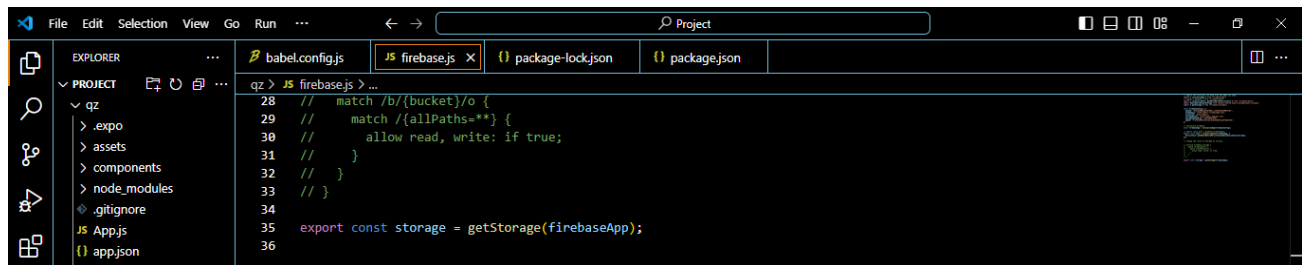
`/>`

- **<Stack.Screen>**: This is a component provided by React Navigation's stack navigator. It defines a screen within a stack navigator.
- **name="PoseDetection"**: This attribute specifies the name of the screen. In this case, the screen is named "PoseDetection". This name is used to navigate to this screen from other parts of the application.
- **component={PoseDetection}**: This attribute specifies the component to be rendered when this screen is navigated to. In this case, it's referring to a component named **PoseDetection**.
- **options={{ title: "Pose Detection" }}**: This attribute defines the options for this screen, including how it appears in the navigation header. In this case, it sets the title of the screen to "Pose Detection", which will be displayed in the navigation bar when this screen is active.

So, when the user navigates to the "PoseDetection" screen in the app, the **PoseDetection** component will be rendered, and the navigation header will display the title "Pose Detection".

Firestore.js:





```
const firebaseConfig = {  
  apiKey: "AIzaSyBWGaoQTq5YUDmL_lwZkXeXbah9MBjwfOI",  
  authDomain: "mlkit-86acc.firebaseio.com",  
  projectId: "mlkit-86acc",  
  storageBucket: "mlkit-86acc.appspot.com",  
  messagingSenderId: "191346997120",  
  appId: "1:191346997120:web:81d3163b9fa5e6f02d7365",  
};
```

- **firebaseConfig:** This is a constant variable (using **const**) that holds the Firebase configuration settings. It's commonly named **firebaseConfig** but can be named anything.
- **apiKey:** This is the API key provided by Firebase for authentication purposes. It's used to authenticate requests from your application to Firebase services.
- **authDomain:** This is the domain of the Firebase authentication service. It's used to authenticate users through Firebase authentication.
- **projectId:** This is the ID of your Firebase project. It's used to identify your project within the Firebase ecosystem.
- **storageBucket:** This is the URL of the Cloud Storage bucket associated with your Firebase project. It's used for storing and retrieving files from Firebase Cloud Storage.
- **messagingSenderId:** This is the sender ID for Firebase Cloud Messaging. It's used for sending push notifications to devices from your Firebase project.
- **appId:** This is the ID of your Firebase web application. It uniquely identifies your application within the Firebase ecosystem.

These configuration settings are crucial for initializing Firebase within application. They enable the app to connect to Firebase services such as authentication, database, storage, and messaging. Make sure to keep these settings secure and not expose them publicly, as they grant access to Firebase project.

Resnet 50 Model:

This code utilizes the “ResNet 50” model provided by the TensorFlow.js library for image classification tasks.

1. Initialization: The code imports the necessary libraries including TensorFlow.js and ResNet model from `@tensorflow-models/resnet`.

2. Permissions and TensorFlow Initialization: It requests permission to access the media library if the app is not running on the web platform. Then, it initializes TensorFlow.js and sets the backend to "rn-webgl".

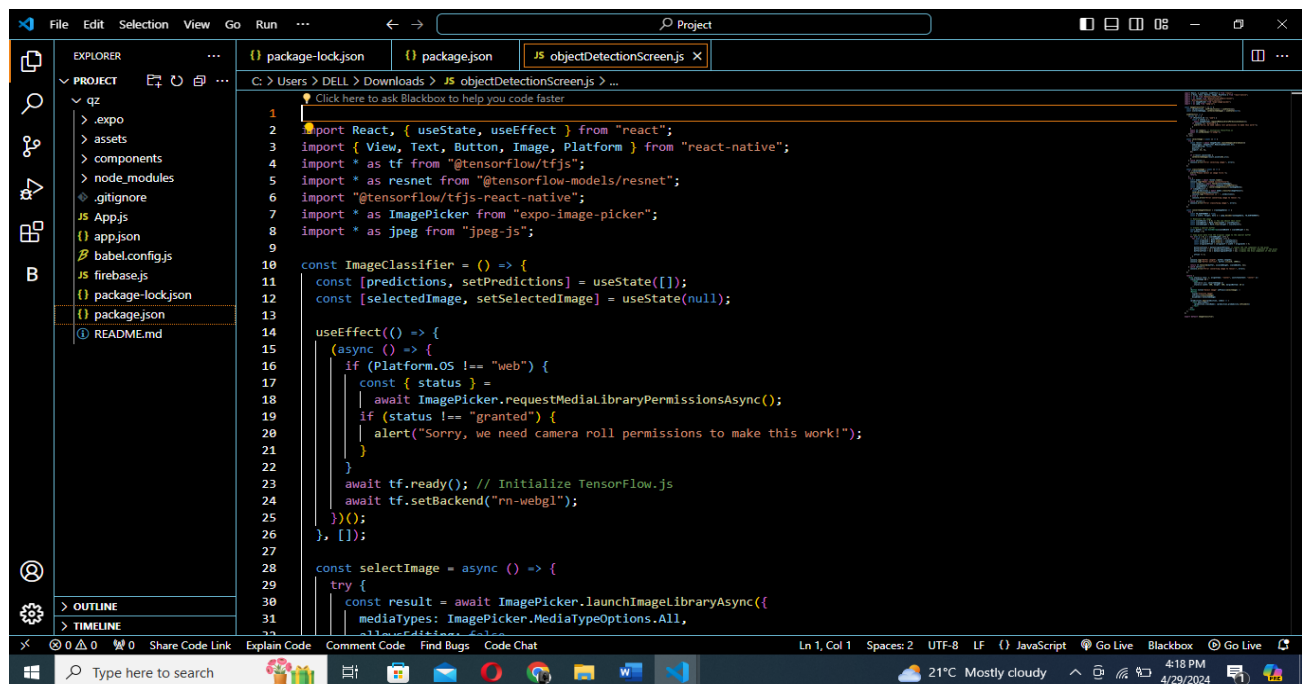
3. Image Selection: It provides a function `selectImage()` to allow users to select an image from their device's media library using Expo's ImagePicker.

4. Image Classification: When the user selects an image and clicks the "Classify Image" button, it loads the ResNet-50 model asynchronously using `resnet.load()`. Then, it fetches the selected image, converts it into a tensor, and passes it to the model for classification.

5. Conversion to Tensor: The function `convertImageToTensor()` converts the raw image data into a tensor suitable for input to the ResNet model. It decodes the image, downscales it, and creates a tensor from the pixel data.

6. Displaying Predictions: Finally, the predictions from the ResNet model are displayed as text on the screen, showing the class names and their corresponding probabilities.

This code demonstrates how to use the ResNet-50 model for image classification in a React Native application, allowing users to select an image, classify it, and display the predicted results.



```
1  import React, { useState, useEffect } from "react";
2  import { View, Text, Button, Image, Platform } from "react-native";
3  import * as tf from "@tensorflow/tfjs";
4  import * as resnet from "@tensorflow-models/resnet";
5  import "@tensorflow/tfjs-react-native";
6  import * as ImagePicker from "expo-image-picker";
7  import * as jpeg from "jpeg-js";
8
9
10 const ImageClassifier = () => {
11   const [predictions, setPredictions] = useState([]);
12   const [selectedImage, setSelectedImage] = useState(null);
13
14   useEffect(() => {
15     (async () => {
16       if (Platform.OS !== "web") {
17         const { status } =
18           await ImagePicker.requestMediaLibraryPermissionsAsync();
19         if (status !== "granted") {
20           alert("Sorry, we need camera roll permissions to make this work!");
21         }
22       }
23       await tf.ready(); // Initialize TensorFlow.js
24       await tf.setBackend("rn-webgl");
25     })();
26   }, []);
27
28   const selectImage = async () => {
29     try {
30       const result = await ImagePicker.launchImageLibraryAsync({
31         mediaTypes: ImagePicker.MediaTypeOptions.All,
32         allowsEditing: false,
```

```
10 const ImageClassifier = () => {
28   const selectImage = async () => {
30     const result = await ImagePicker.launchImageLibraryAsync({
31       mediaTypes: ImagePicker.MediaTypeOptions.All,
32       allowsEditing: false,
33       quality: 1,
34       aspect: [4, 3],
35     });
36   };
37   if (!result.cancelled) {
38     | setSelectedImage(result.assets[0].uri);
39   }
40 } catch (error) {
41   console.error("Error selecting image:", error);
42 }
43 };
44
45 const classifyImage = async () => {
46   if (!selectedImage) {
47     alert("Please select an image first.");
48     return;
49   }
50   try {
51     const model = await resnet.load();
52     console.log("Model loaded successfully");
53     const response = await fetch(selectedImage);
54     const rawImageData = await response.arrayBuffer();
55     const imageTensor = convertImageToTensor(rawImageData);
56     if (imageTensor) {
57       const predictions = await model.classify(imageTensor);
58       setPredictions(predictions);
59       console.log("Prediction is : ", predictions);
```

```
45   const classifyImage = async () => {
57     const predictions = await model.classify(imageTensor);
58     setPredictions(predictions);
59     console.log("Prediction is : ", predictions);
60   } else {
61     console.error("Error converting image to tensor.");
62   }
63 } catch (error) {
64   console.error("Error classifying image:", error);
65 }
66 };
67
68 const convertImageToTensor = (rawImageData) => {
69   try {
70     const TO_UINT8ARRAY = true;
71     const { width, height, data } = jpeg.decode(rawImageData, TO_UINT8ARRAY);
72
73     // Downscale the image
74     const scaleFactor = 0.5; // You can adjust this value
75     const scaledWidth = Math.floor(width * scaleFactor);
76     const scaledHeight = Math.floor(height * scaleFactor);
77
78     // Create a smaller buffer
79     const buffer = new Uint8Array(scaledWidth * scaledHeight * 3);
80     let offset = 0;
81
82     // Copy pixel data from the original image to the smaller buffer
83     for (let y = 0; y < scaledHeight; y++) {
84       for (let x = 0; x < scaledWidth; x++) {
85         const originalX = Math.floor(x / scaleFactor);
86         const originalY = Math.floor(y / scaleFactor);
```

```
10 const ImageClassifier = () => {
68   const convertImageToTensor = (rawImageData) => {
85     const originalX = Math.floor(x / scaleFactor);
86     const originalY = Math.floor(y / scaleFactor);
87     const originalOffset = (originalY * width + originalX) * 4;
88
89     buffer[offset] = data[originalOffset]; // copies the red component of the pixel
90     buffer[offset + 1] = data[originalOffset + 1]; //copies the green component of the pixel
91     buffer[offset + 2] = data[originalOffset + 2]; //copies the blue component of the pixel
92
93     offset += 3;
94   }
95 }
96 console.log("Buffer Length", buffer.length);
97 console.log("Buffer entries", buffer.slice(0, 1000));
98
99 return tf.tensor3d(buffer, [scaledHeight, scaledWidth, 3]);
100 } catch (error) {
101   console.error("Error converting image to tensor:", error);
102 }
103 };
104
105 return (
106   <View style={{ flex: 1, alignItems: "center", justifyContent: "center" }}>
107     {selectedImage && (
108       <Image
109         source={{ uri: selectedImage }}
110         style={{ width: 300, height: 300, marginBottom: 20 }}
111       />
112     )}
113     <Button title="Select Image" onPress={selectImage} />
114   </View>
115 );
```

```
111 //
112 })
113 <Button title="Select Image" onPress={selectImage} />
114 <Button
115   title="Classify Image"
116   onPress={classifyImage}
117   disabled={!selectedImage}
118 />
119 {predictions.map((prediction, index) => (
120   <Text key={index}>
121     {prediction.className}: {prediction.probability.toFixed(3)}
122   </Text>
123 )
124 )}
125 </View>
126 );
127
128 export default ImageClassifier;
129
130
```

Output:

