

MOBILE APPLICATION DEVELOPMENT



QUIZ 02

Submitted by:

Muhammad Hammad(21-SE-30)

Submitted to:

Mam Kanwal

Semester:

Sixth

Date of Submission:

27th -April – 2024

Software Engineering Department
University of Engineering and Technology
Taxila

Question 01:

In this assignment, you will be working with the codebase from Lecture 10-12 of the React Native 2024 course, accessible at the GitHub repository: <https://github.com/KanYousaf/Lecture-10-11-12-ReactNative2024>. Your task involves enhancing the user interface (UI) of an existing application and integrating an AI model of your choice. Additionally, you'll implement the AI model within the **objectDetectionScreen.js** file, leveraging the provided codebase.

Solution

Project selected:

Object Detection with TensorFlow and mobileNet model

GitHub link :

https://github.com/3MuHaMm9AdHaMmAd6/ObjectDetectionSystem_MadQuiz02.git

1. Code Reuse and Cleanup:

The areas relevant to UI enhancement and menu integration were :

- a. The section of the app that dealt with taking the image through camera.
- b. The section of the app that dealt with allowing user to select image to be processed from their device
- c. The section that allowed the user to view the history, with most recent image , and the identified result showing at the top.
- d. The landing page that gave an introduction about the app

All these sections are to be integrated through help of bottom tab navigator and stack navigator where bottom tab navigator will provide navigation across sections and stack navigator will provide navigation within sections.

To Remove any redundant or unused code to ensure code cleanliness and efficiency.

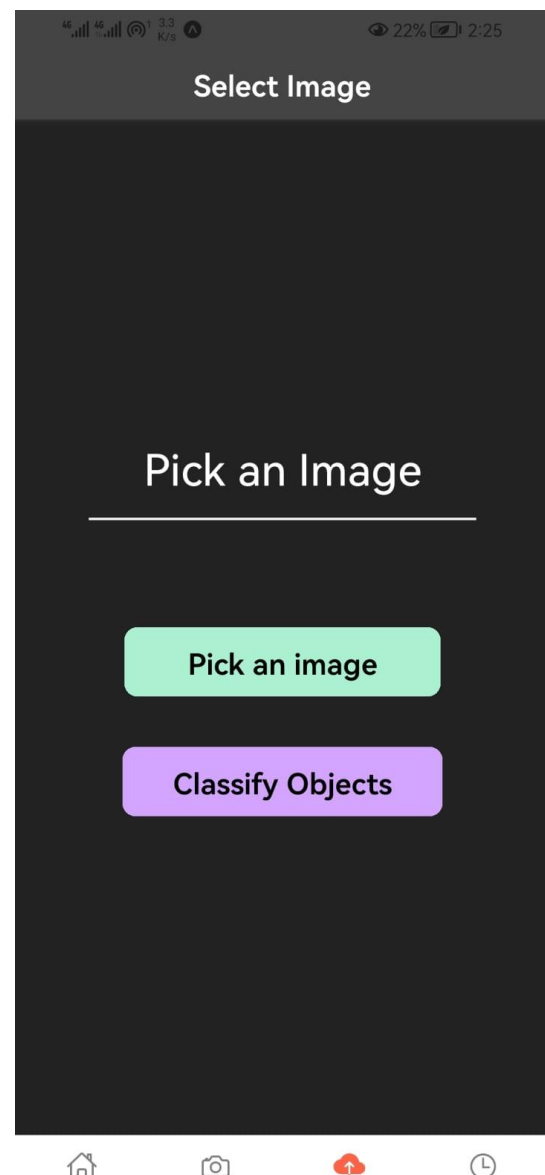
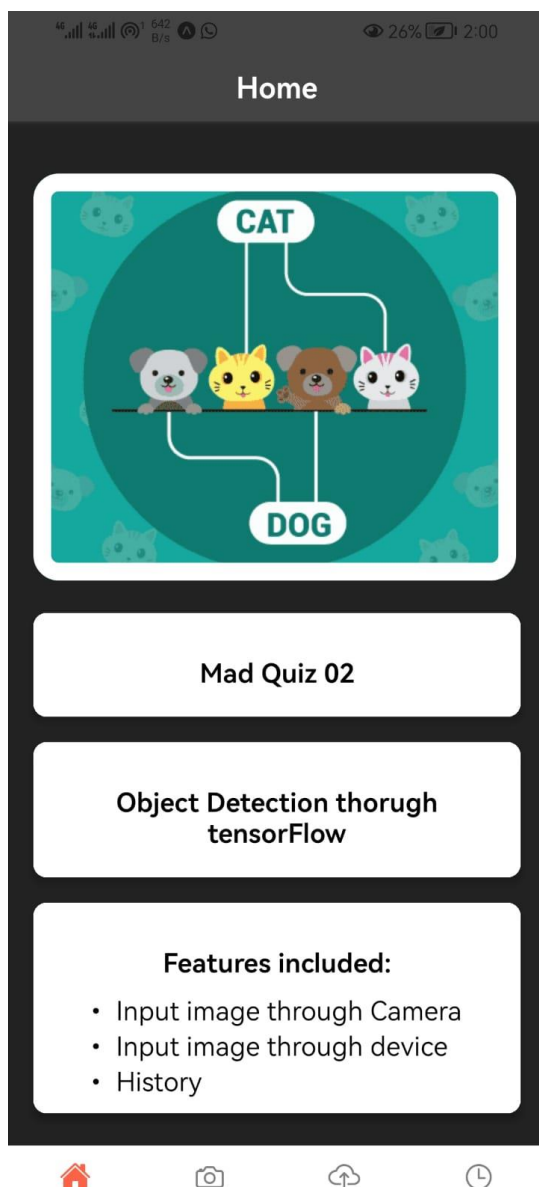
- a) The code that was not useful was commented out and their imports were also commented out
- b) Some files had to be re written to for the purpose of providing features required in the current circumstances.

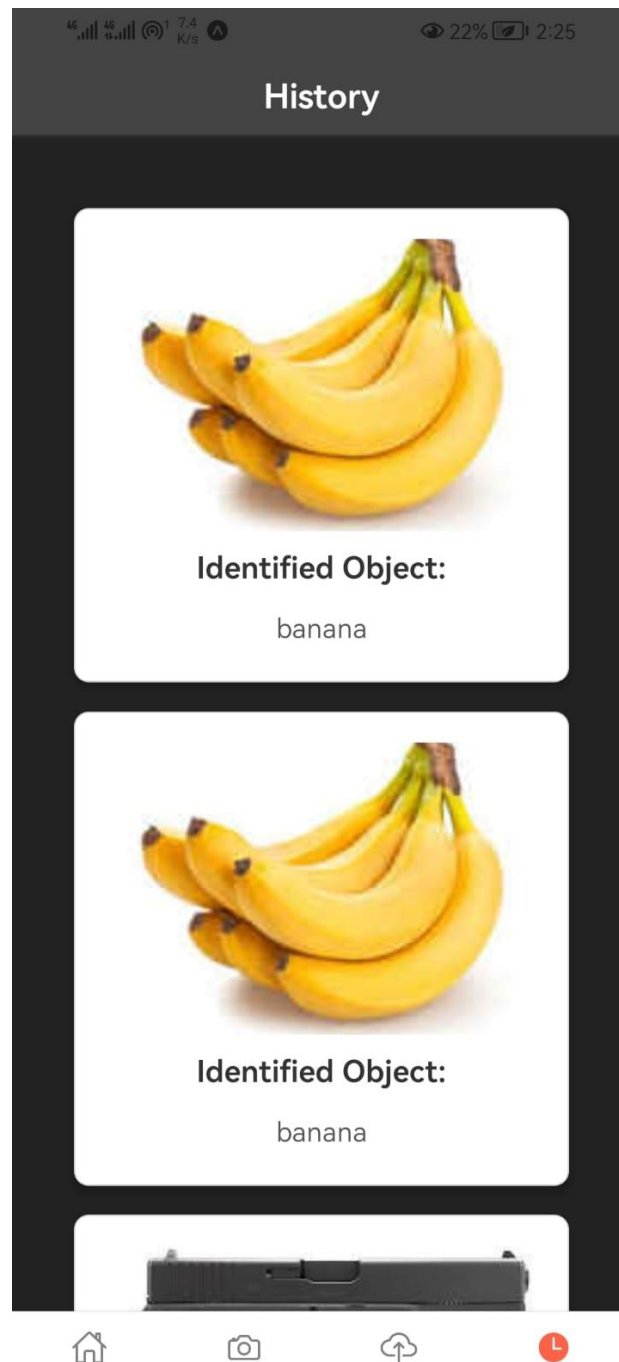
2. UI Enhancement:

Following measures were taken to enhance the UI

- In each screen, decent colors were used which provided a good look to the app
- Simple UI was preferred, to provide user with seamless experience
- Proper padding and spacing was ensured to prevent text from being obstructed or controls being rammed into each other, saving the user from being feeling confused.
- Each section was labeled to indicate the functionality of that section and the information it contained and controls were also labeled to make user aware of the action they were going to execute.

Below are images of some section within the app :





3. Menu Integration:

Bottom Tab Navigator was used in combination with the stack navigator. Bottom Tab Navigator provide navigation along following four sections:

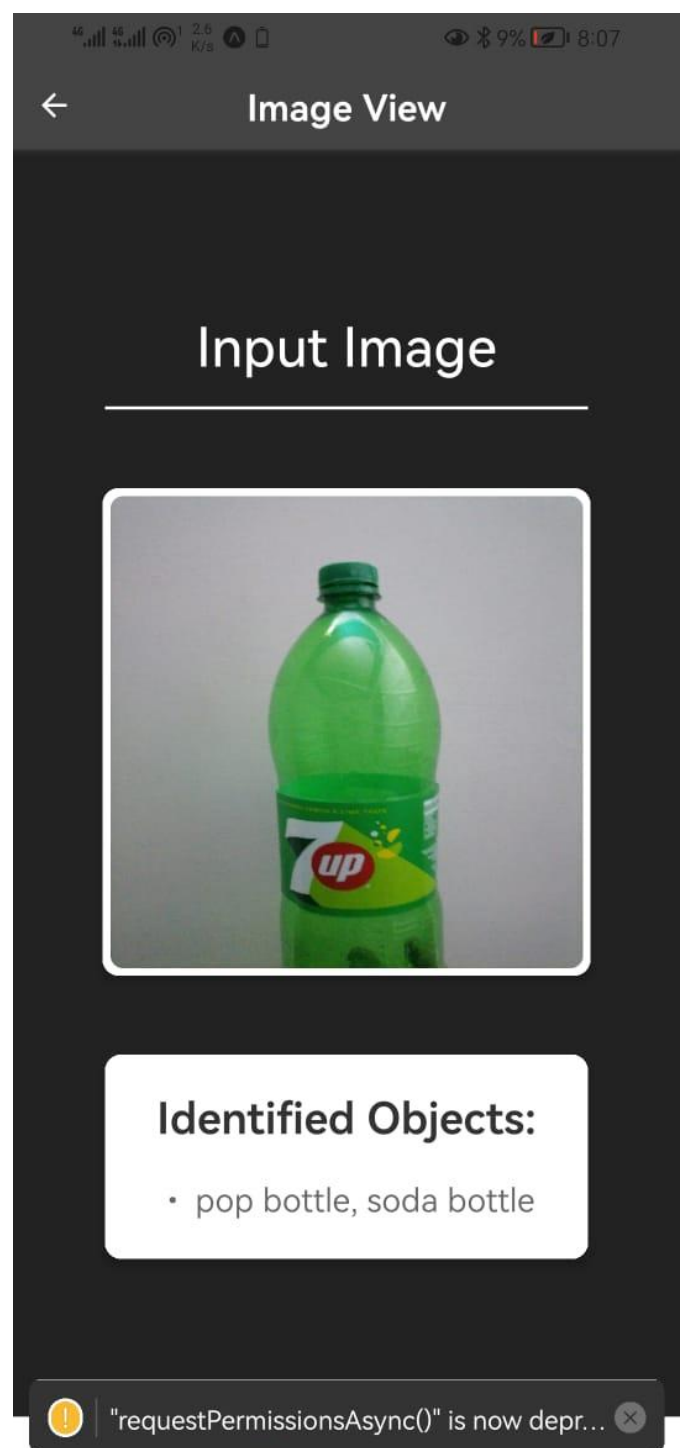
- a. Home Section (welcome page to the app)
- b. Camera Section (Allows input to be given to model through the camera)
- c. Pick Image Section (Allows input to be given to model through the storage on device)

- d. History (stores the image input given along with the output generated by the model)

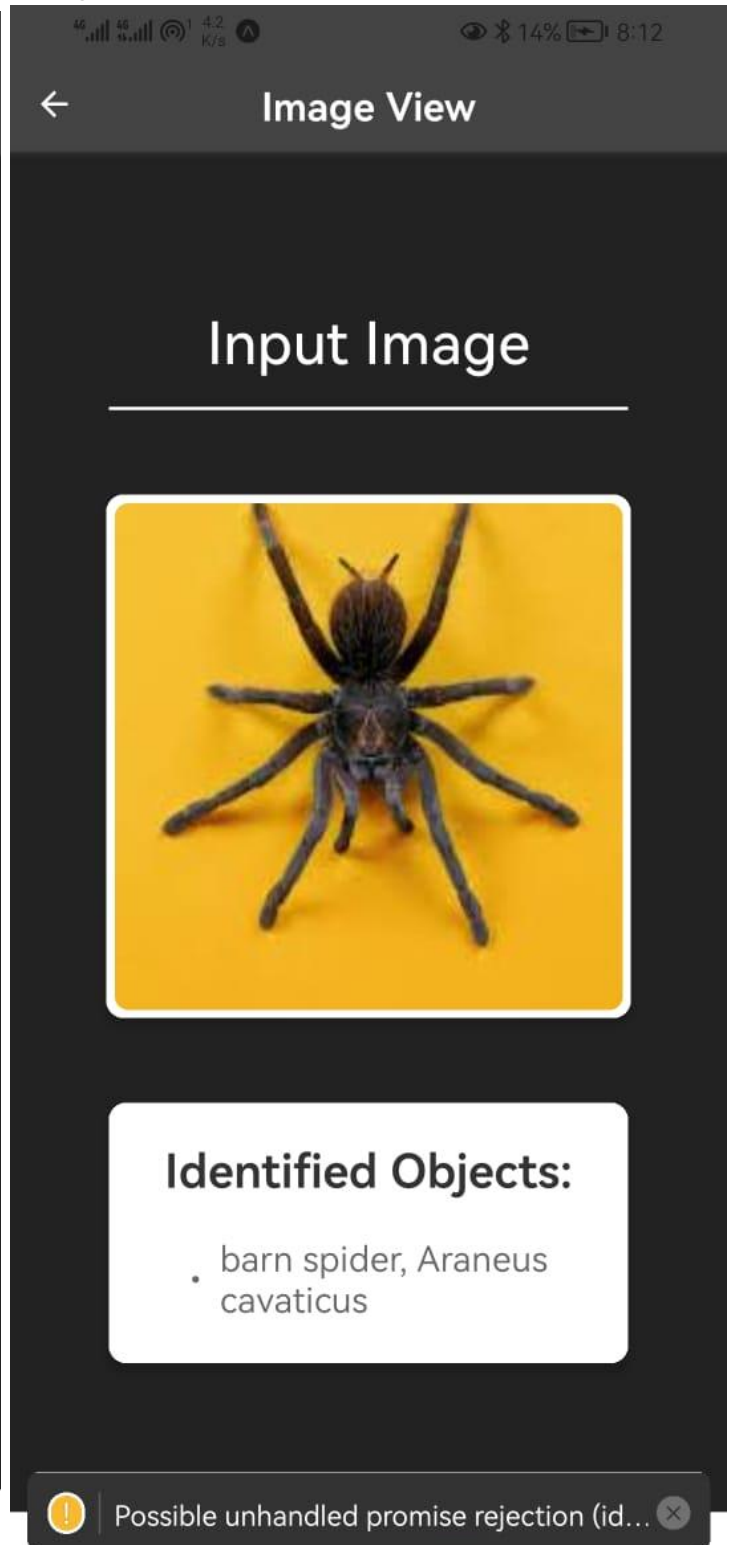
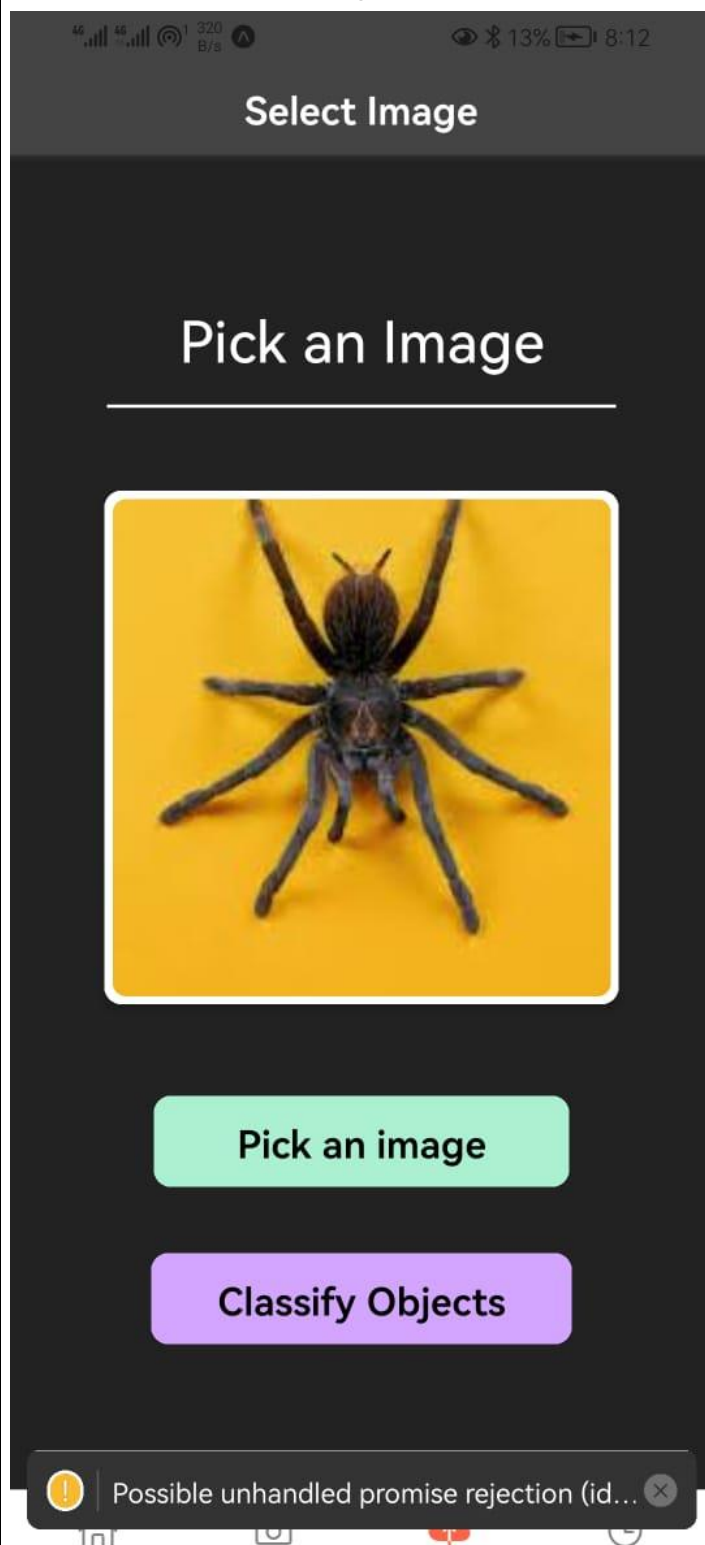
The stack navigators provided navigation across the following section

- a) The 'TakeImageSections' uses stack navigator to provide navigation to result , after user had captured the image
- b) The 'SelectImageFromDevice' uses stack navigator to provide navigation to result, after user has selected the image from device.

The "TakeImageSections" has following two screens:



The “SelectImageFromDevice” has following two screens:

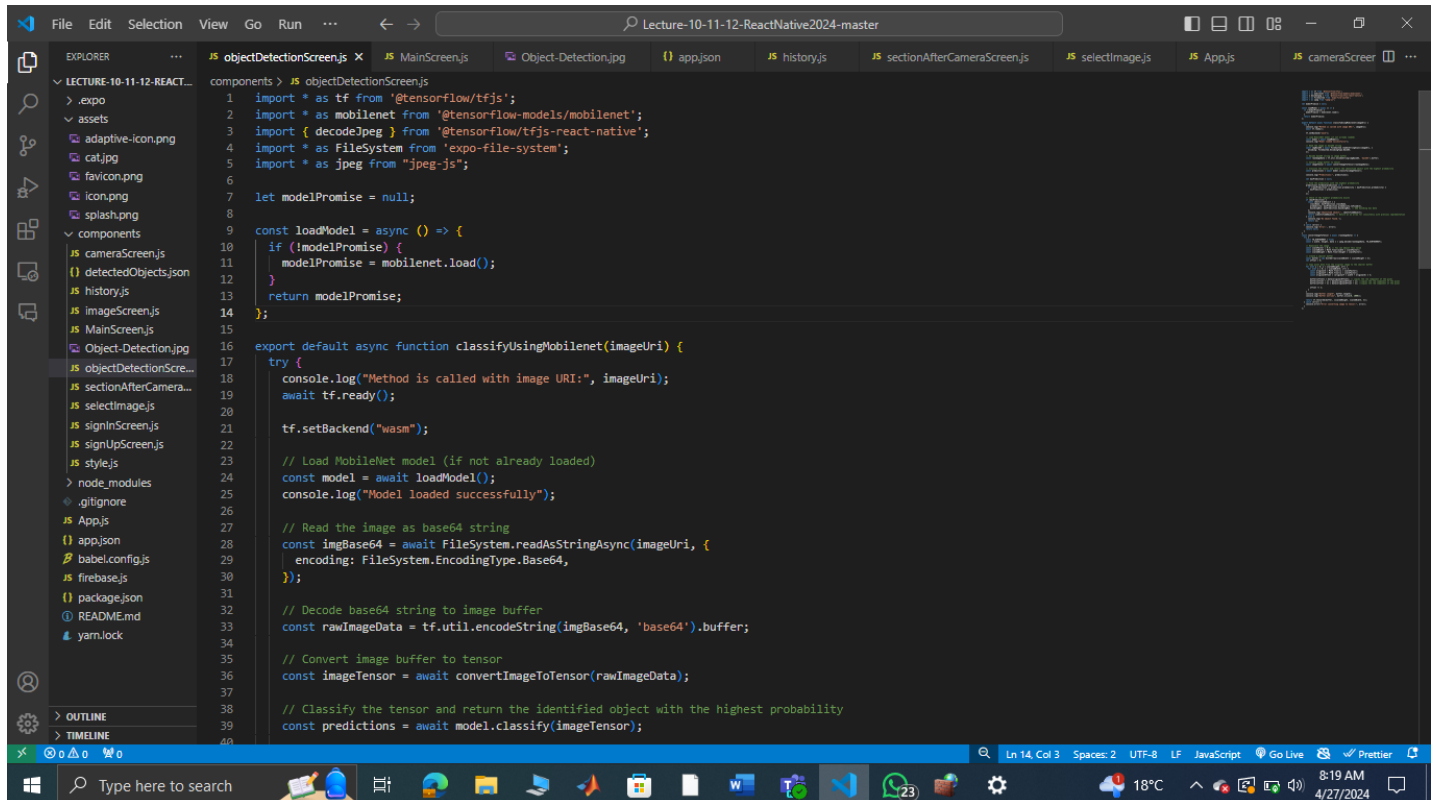


4. AI Model Integration:

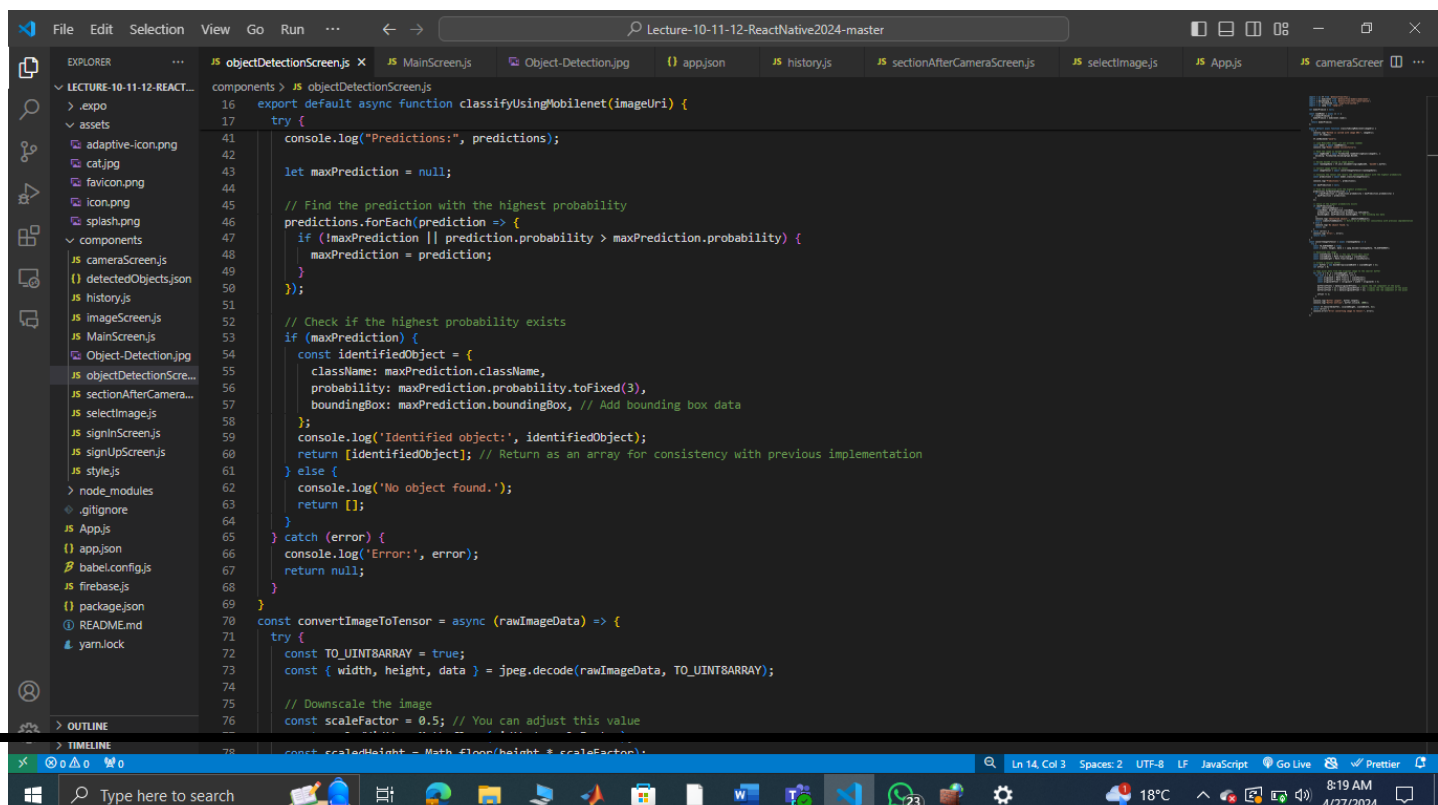
Model of choice:

Mobile net model of Tensor flow

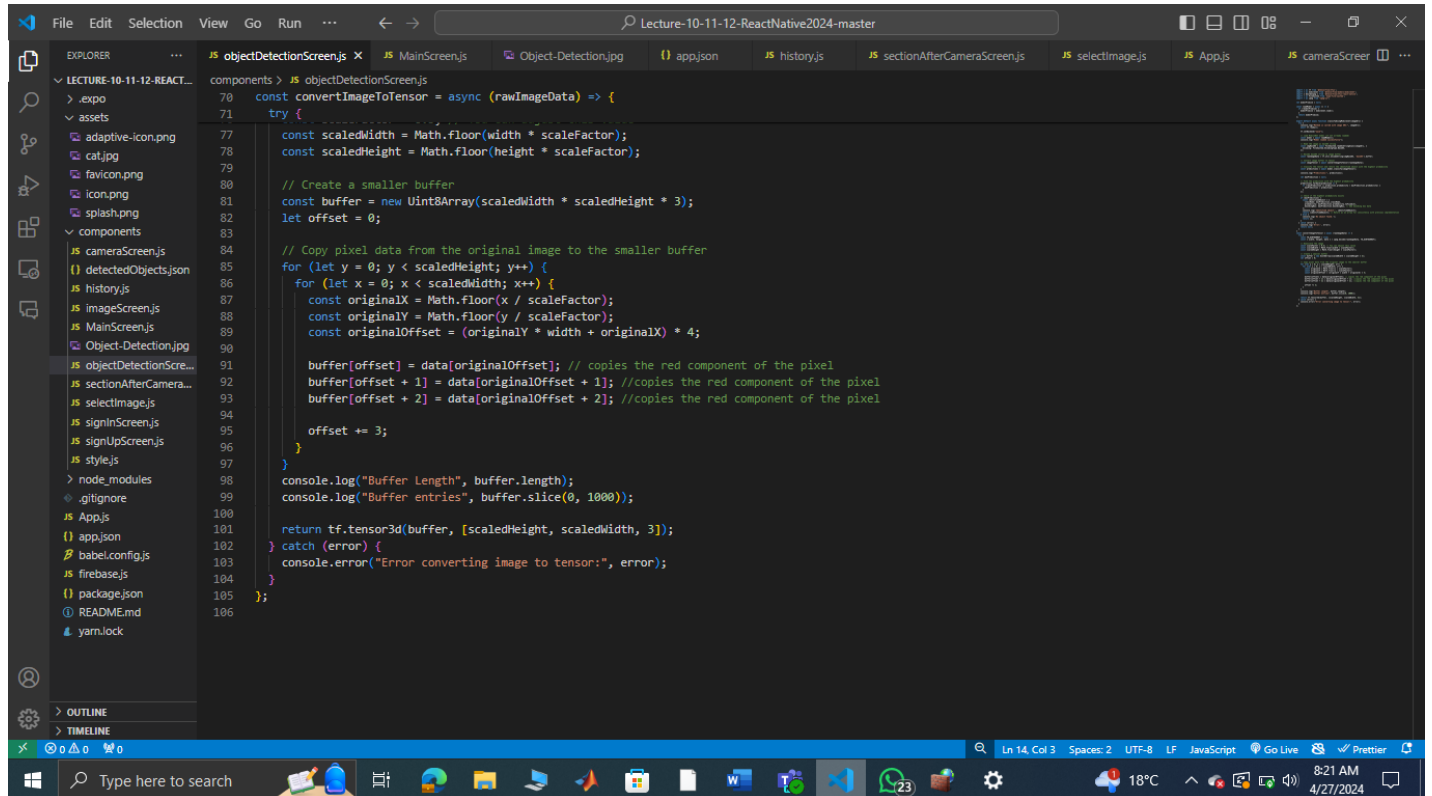
Code:



```
1 import * as tf from '@tensorflow/tfjs';
2 import * as mobilenet from '@tensorflow-models/mobilenet';
3 import { decodeJpeg } from '@tensorflow/tfjs-react-native';
4 import * as FileSystem from 'expo-file-system';
5 import * as jpeg from 'jpeg-js';
6
7 let modelPromise = null;
8
9 const loadModel = async () => {
10   if (!modelPromise) {
11     modelPromise = mobilenet.load();
12   }
13   return modelPromise;
14 };
15
16 export default async function classifyUsingMobileNet(imageUri) {
17   try {
18     console.log("Method is called with image URI:", imageUri);
19     await tf.ready();
20
21     tf.setBackend("wasm");
22
23     // Load MobileNet model (if not already loaded)
24     const model = await loadModel();
25     console.log("Model loaded successfully");
26
27     // Read the image as base64 string
28     const imgBase64 = await FileSystem.readAsStringAsync(imageUri, {
29       encoding: FileSystem.EncodingType.Base64,
30     });
31
32     // Decode base64 string to image buffer
33     const rawImageData = tf.util.encodeString(imgBase64, 'base64').buffer;
34
35     // Convert image buffer to tensor
36     const imageTensor = await convertImageToTensor(rawImageData);
37
38     // Classify the tensor and return the identified object with the highest probability
39     const predictions = await model.classify(imageTensor);
```



```
40
41   console.log("Predictions:", predictions);
42
43   let maxPrediction = null;
44
45   // Find the prediction with the highest probability
46   predictions.forEach(prediction => {
47     if (!maxPrediction || prediction.probability > maxPrediction.probability) {
48       maxPrediction = prediction;
49     }
50   });
51
52   // Check if the highest probability exists
53   if (maxPrediction) {
54     const identifiedObject = {
55       className: maxPrediction.className,
56       probability: maxPrediction.probability.toFixed(3),
57       boundingBox: maxPrediction.boundingBox, // Add bounding box data
58     };
59     console.log('Identified object:', identifiedObject);
60     return [identifiedObject]; // Return as an array for consistency with previous implementation
61   } else {
62     console.log('No object found.');
```

```
70 const convertImageToTensor = async (rawImageData) => {
71   try {
72     const scaledWidth = Math.floor(width * scaleFactor);
73     const scaledHeight = Math.floor(height * scaleFactor);
74
75     // Create a smaller buffer
76     const buffer = new Uint8Array(scaledWidth * scaledHeight * 3);
77     let offset = 0;
78
79     // Copy pixel data from the original image to the smaller buffer
80     for (let y = 0; y < scaledHeight; y++) {
81       for (let x = 0; x < scaledWidth; x++) {
82         const originalX = Math.floor(x / scaleFactor);
83         const originalY = Math.floor(y / scaleFactor);
84         const originalOffset = (originalY * width + originalX) * 4;
85
86         buffer[offset] = data[originalOffset]; // copies the red component of the pixel
87         buffer[offset + 1] = data[originalOffset + 1]; //copies the red component of the pixel
88         buffer[offset + 2] = data[originalOffset + 2]; //copies the red component of the pixel
89
90         offset += 3;
91       }
92     }
93
94     console.log("Buffer Length", buffer.length);
95     console.log("Buffer entries", buffer.slice(0, 1000));
96
97     return tf.tensor3d(buffer, [scaledHeight, scaledWidth, 3]);
98   } catch (error) {
99     console.error("Error converting image to tensor:", error);
100   }
101 }
```

Operation:

1. Loading the Model:

The application loads the MobileNet model asynchronously using the `mobilenet.load()` function. This ensures that the model is loaded only once and reused for subsequent classifications.

2. Image Processing:

The image provided by the user is read as a base64 string using the Expo File System API. The base64 string is decoded into raw image data. The raw image data is converted into a tensor, which is compatible with TensorFlow.js operations. During image conversion, the image is downscaled to reduce computational complexity while preserving essential features.

3. Classification:

The converted image tensor is fed into the MobileNet model for classification. The model predicts the probability distribution over a set

of pre-defined classes. The class with the highest probability is considered the identified object.

4. Result Presentation:

The identified object's class name, probability, and bounding box (if available) are logged for debugging purposes. If an object is successfully identified, the result is returned as an array containing the object's details. If no object is found or an error occurs during the process, appropriate messages are logged, and an empty array or null is returned.