



# React Native Project with NodeJS and MongoDB- Part 2

Nabendu Biswas · [Follow](#)

8 min read · Apr 27, 2022



16





React Native

Welcome to the part-2 of the series and we will start where we left. You can find part-1 [here](#).

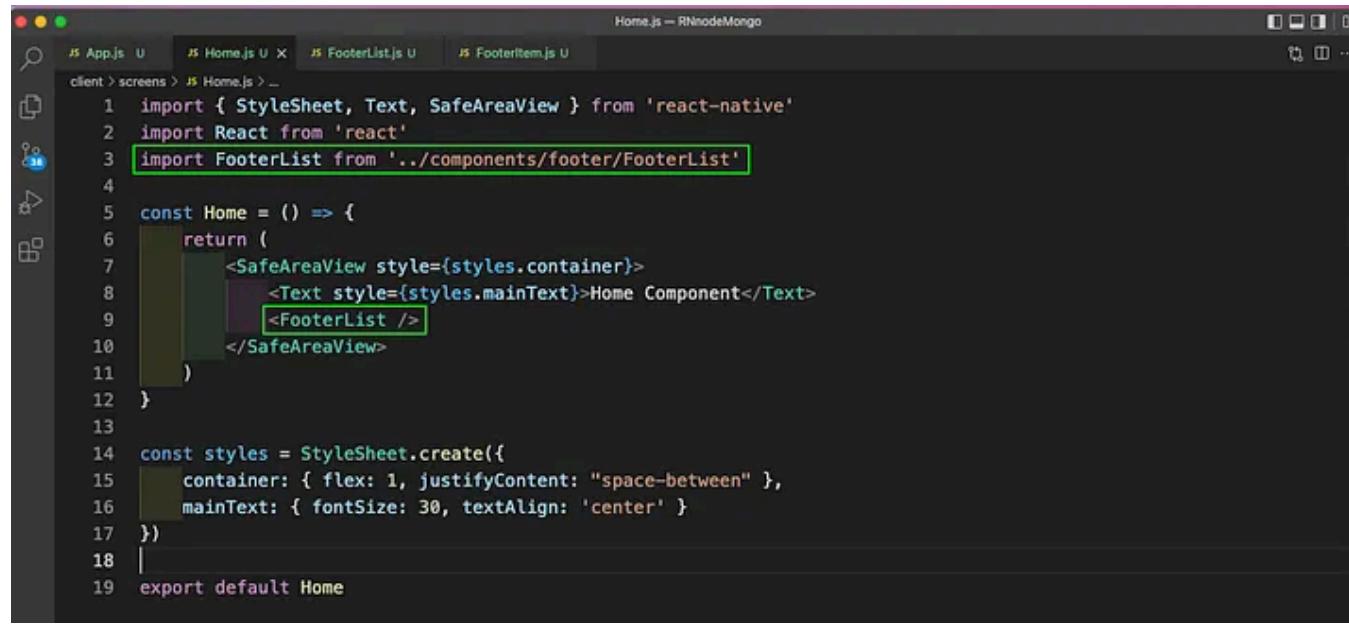
We will first install react native vector icons in our project.

```
nabendubiswas@Nabendus-Mac-mini ~/Projects/RNnodeMongo/client (master)$ npm i react-native-vector-icons
added 10 packages, and audited 921 packages in 5s
34 packages are looking for funding
  run `npm fund` for details
5 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
nabendubiswas@Nabendus-Mac-mini ~/Projects/RNnodeMongo/client (master)$ []
```

After that include a **FooterList** component in the **Home.js** file.



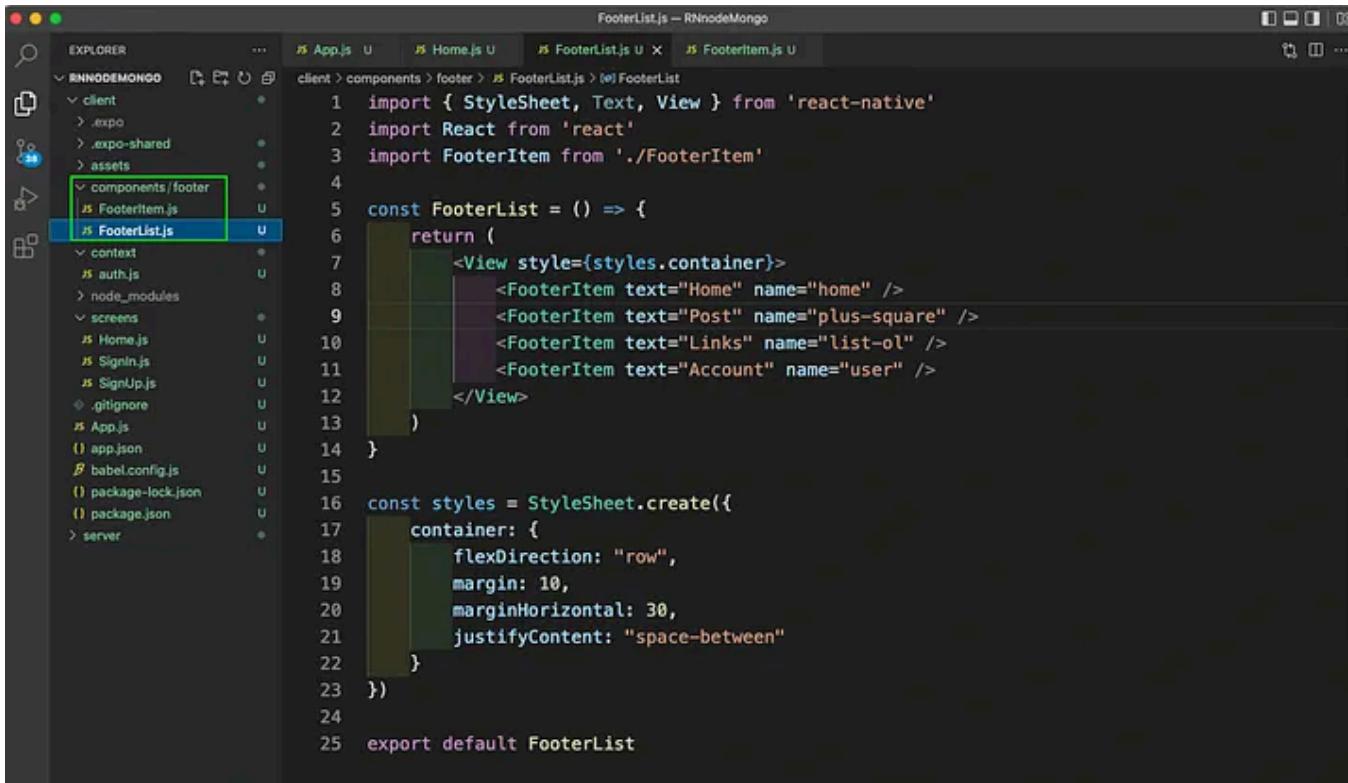
The screenshot shows a code editor window with the title "Home.js – RNNodeMongo". The editor displays the following code:

```
client > screens > Home.js > ...
1 import { StyleSheet, Text, SafeAreaView } from 'react-native'
2 import React from 'react'
3 import FooterList from '../components/footer/FooterList'
4
5 const Home = () => {
6   return (
7     <SafeAreaView style={styles.container}>
8       <Text style={styles.mainText}>Home Component</Text>
9       <FooterList />
10    </SafeAreaView>
11  )
12}
13
14 const styles = StyleSheet.create({
15   container: { flex: 1, justifyContent: "space-between" },
16   mainText: { fontSize: 30, textAlign: 'center' }
17 })
18
19 export default Home
```

Home.js

Next, create a **footer** folder in the **components** folder and a file **FooterList.js** inside it. Add the below content in it.

Here, we are sending the **text** and the **name** props to a **FooterItem** component.

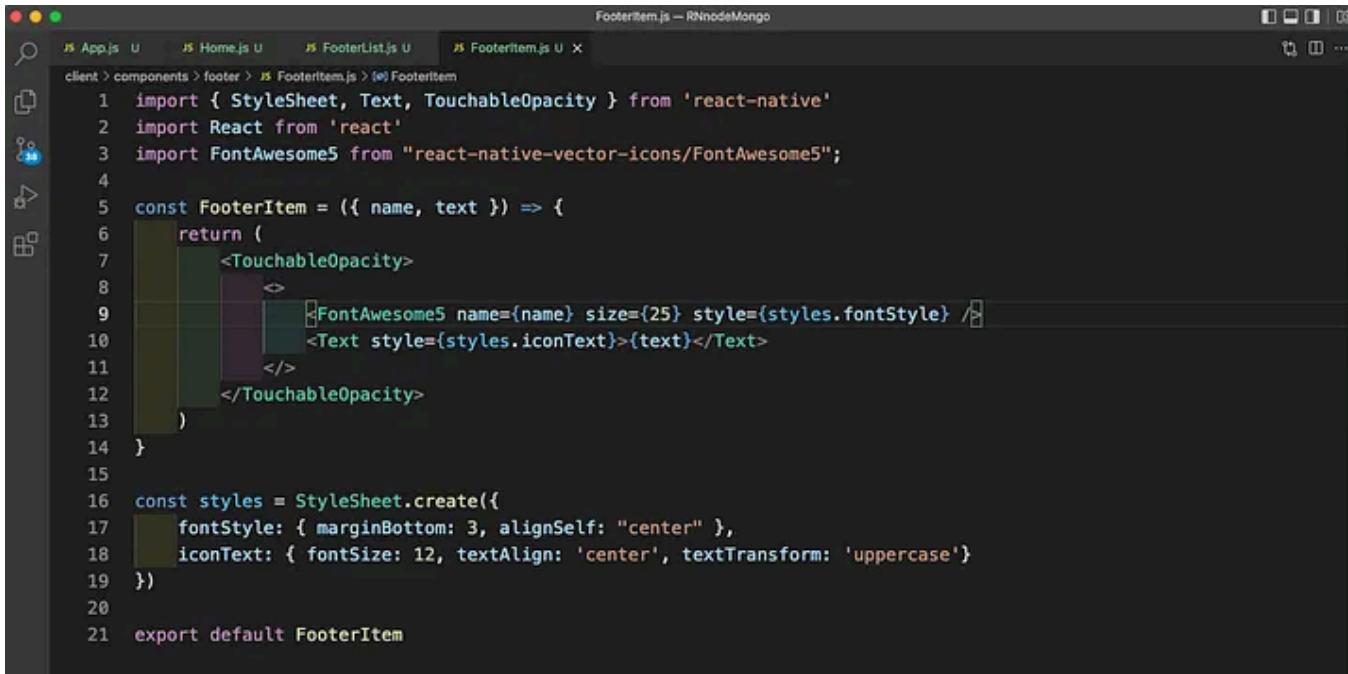


The screenshot shows a dark-themed code editor in VS Code. The left sidebar displays a project structure under 'client'. Inside 'client', there are 'components', 'context', 'screens', and 'server' folders. Within 'components', there is a 'footer' folder containing two files: 'FooterItem.js' and 'FooterList.js'. The 'FooterList.js' file is currently selected and highlighted with a blue bar at the top of its code area. The code itself is a functional component named 'FooterList' that returns a horizontal view of four 'FooterItem' components. It also defines a style sheet named 'styles' with a single 'container' rule.

```
1 import { StyleSheet, Text, View } from 'react-native'
2 import React from 'react'
3 import FooterItem from './FooterItem'
4
5 const FooterList = () => {
6   return (
7     <View style={styles.container}>
8       <FooterItem text="Home" name="home" />
9       <FooterItem text="Post" name="plus-square" />
10      <FooterItem text="Links" name="list-ul" />
11      <FooterItem text="Account" name="user" />
12    </View>
13  }
14
15  const styles = StyleSheet.create({
16    container: {
17      flexDirection: "row",
18      margin: 10,
19      marginHorizontal: 30,
20      justifyContent: "space-between"
21    }
22  })
23
24
25 export default FooterList
```

FooterList.js

Now, create a **FooterItem.js** file inside the footer folder. Here, we are using the FontAwesome5 icons with the passed props.

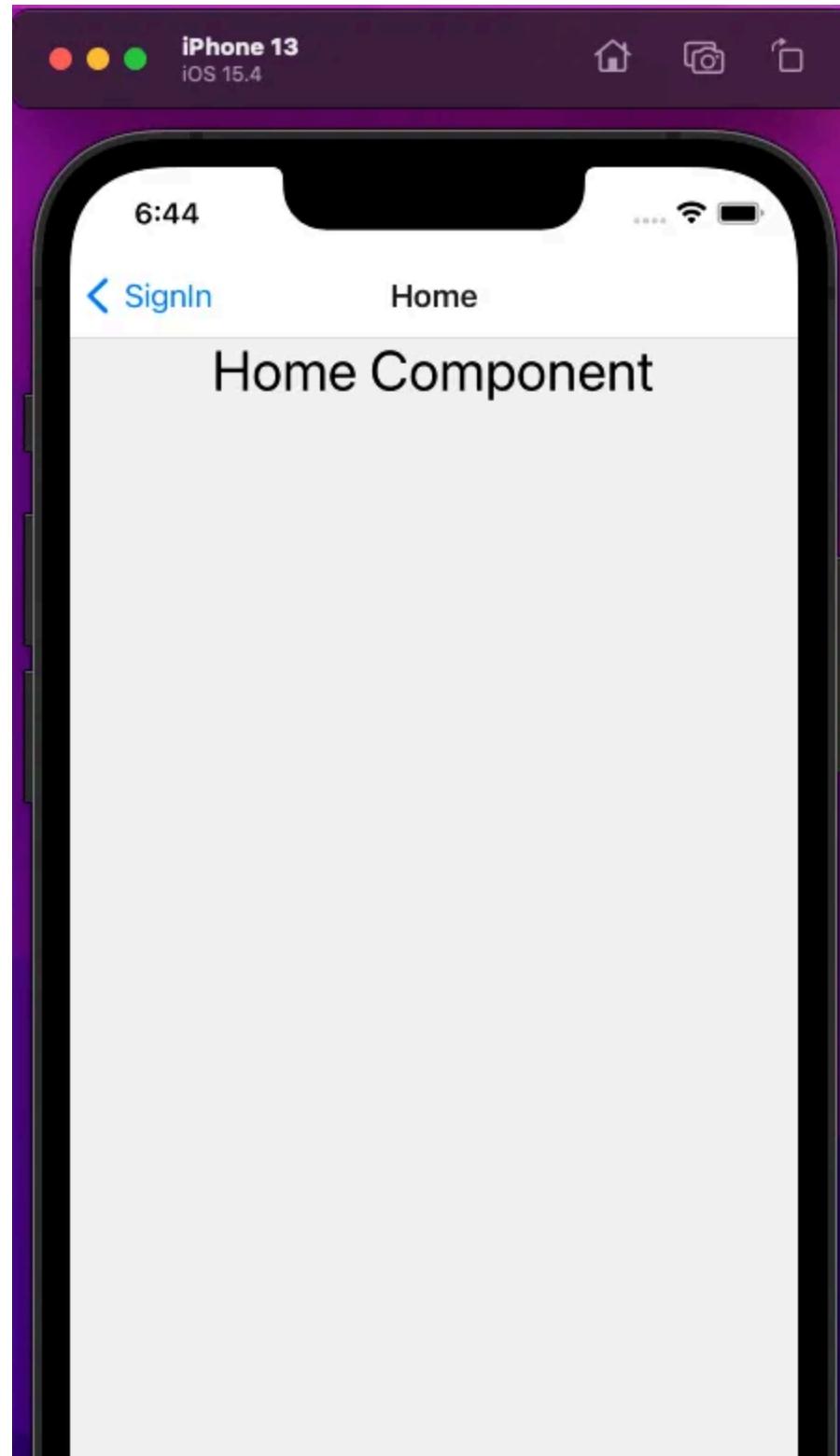


A screenshot of a code editor window titled "FooterItem.js – RNnodeMongo". The window shows the following code:

```
client > components > footer > FooterItem.js > FooterItem
1 import { StyleSheet, Text, TouchableOpacity } from 'react-native'
2 import React from 'react'
3 import FontAwesome5 from "react-native-vector-icons/FontAwesome5";
4
5 const FooterItem = ({ name, text }) => {
6   return (
7     <TouchableOpacity>
8       <>
9         <FontAwesome5 name={name} size={25} style={styles.fontStyle} />
10        <Text style={styles.iconText}>{text}</Text>
11      </>
12     </TouchableOpacity>
13   )
14 }
15
16 const styles = StyleSheet.create({
17   fontStyle: { marginBottom: 3, alignSelf: "center" },
18   iconText: { fontSize: 12, textAlign: 'center', textTransform: 'uppercase' }
19 })
20
21 export default FooterItem
```

FooterItem.js

Now, in our Home component we will see nice icons in footer.





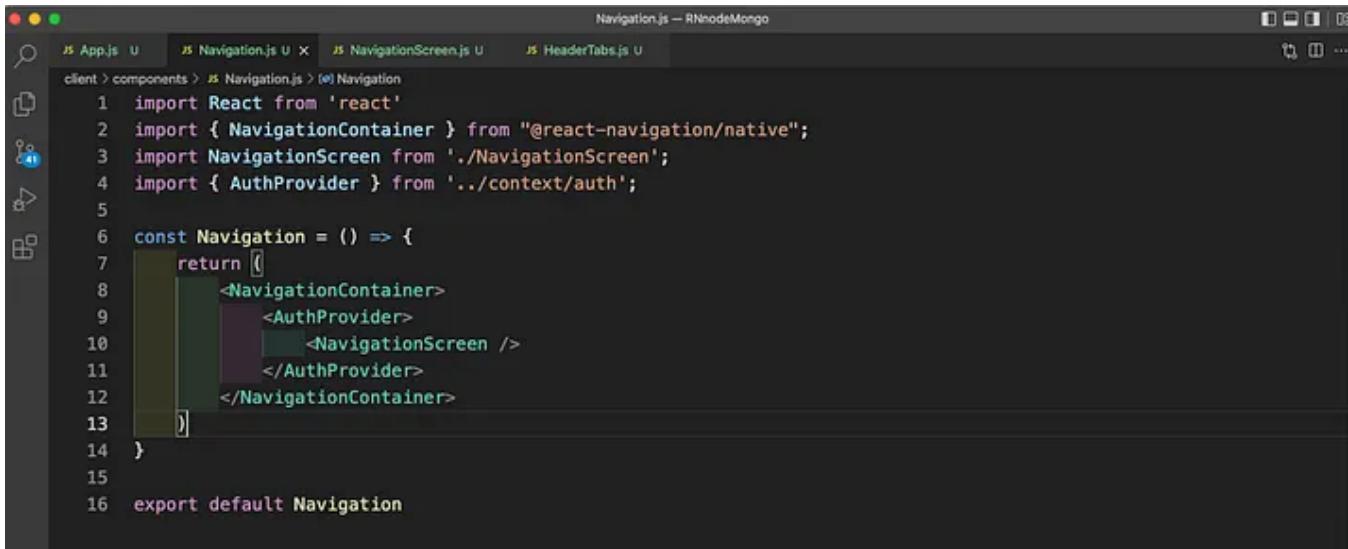
Now, we will refactor our code by moving all content from **App.js** file. Here, we have inserted a **Navigation** component.

```
client > JS App.js U X JS Navigation.js U JS NavigationScreen.js U JS HeaderTabs.js U
JS App.js > ...
1 import React from "react";
2 import Navigation from "./components/Navigation";
3
4 export default function App() {
5   return (
6     <Navigation />
7   )
8 }
9
```

App.js

Next create a **Navigation.js** file inside the components folder and add the below code in it.

Here, we are calling a **NavigationScreen** component.



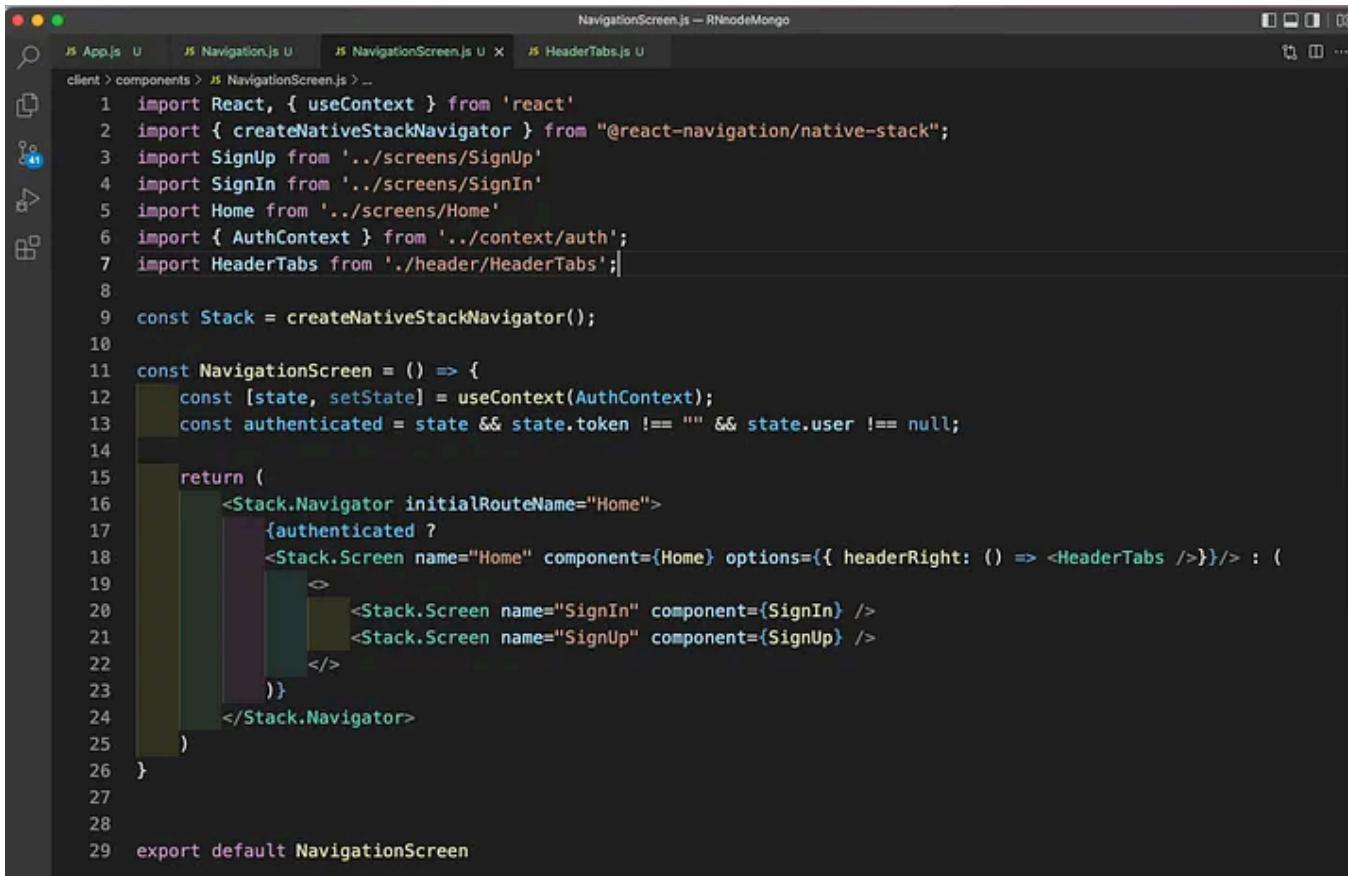
```
Navigation.js — RNnodeMongo
JS App.js U JS Navigation.js U X JS NavigationScreen.js U JS HeaderTabs.js U
client > components > JS Navigation.js > (e) Navigation
1 import React from 'react'
2 import { NavigationContainer } from '@react-navigation/native';
3 import NavigationScreen from './NavigationScreen';
4 import { AuthProvider } from '../context/auth';
5
6 const Navigation = () => {
7   return [
8     <NavigationContainer>
9       <AuthProvider>
10         <NavigationScreen />
11       </AuthProvider>
12     </NavigationContainer>
13   ]
14 }
15
16 export default Navigation
```

Navigation.js

In the **NavigationScreen.js** file we have most of the code from **App.js** file. But we have also added the concept of authentication.

Here, we are checking whether we have the token and if we have it then showing the Home Screen or else showing the SignIn component.

We have also included a new **HeaderTabs** in the Home Screen.



The screenshot shows a code editor window with the title "NavigationScreen.js - RNnodeMongo". The code editor displays the following JavaScript code:

```
JS App.js U JS Navigation.js U JS NavigationScreen.js U X JS HeaderTabs.js U
Client > components > JS NavigationScreen.js > ...
1 import React, { useContext } from 'react'
2 import { createNativeStackNavigator } from "@react-navigation/native-stack";
3 import SignUp from '../screens/SignUp'
4 import SignIn from '../screens/SignIn'
5 import Home from '../screens/Home'
6 import { AuthContext } from '../context/auth';
7 import HeaderTabs from './header/HeaderTabs';
8
9 const Stack = createNativeStackNavigator();
10
11 const NavigationScreen = () => {
12   const [state, setState] = useContext(AuthContext);
13   const authenticated = state && state.token !== "" && state.user !== null;
14
15   return (
16     <Stack.Navigator initialRouteName="Home">
17       {authenticated ? (
18         <Stack.Screen name="Home" component={Home} options={{ headerRight: () => <HeaderTabs /> }}/> : (
19           <>
20             <Stack.Screen name="SignIn" component={SignIn} />
21             <Stack.Screen name="SignUp" component={SignUp} />
22           </>
23         )
24       </Stack.Navigator>
25     )
26   }
27
28
29 export default NavigationScreen
```

NavigationScreen.js

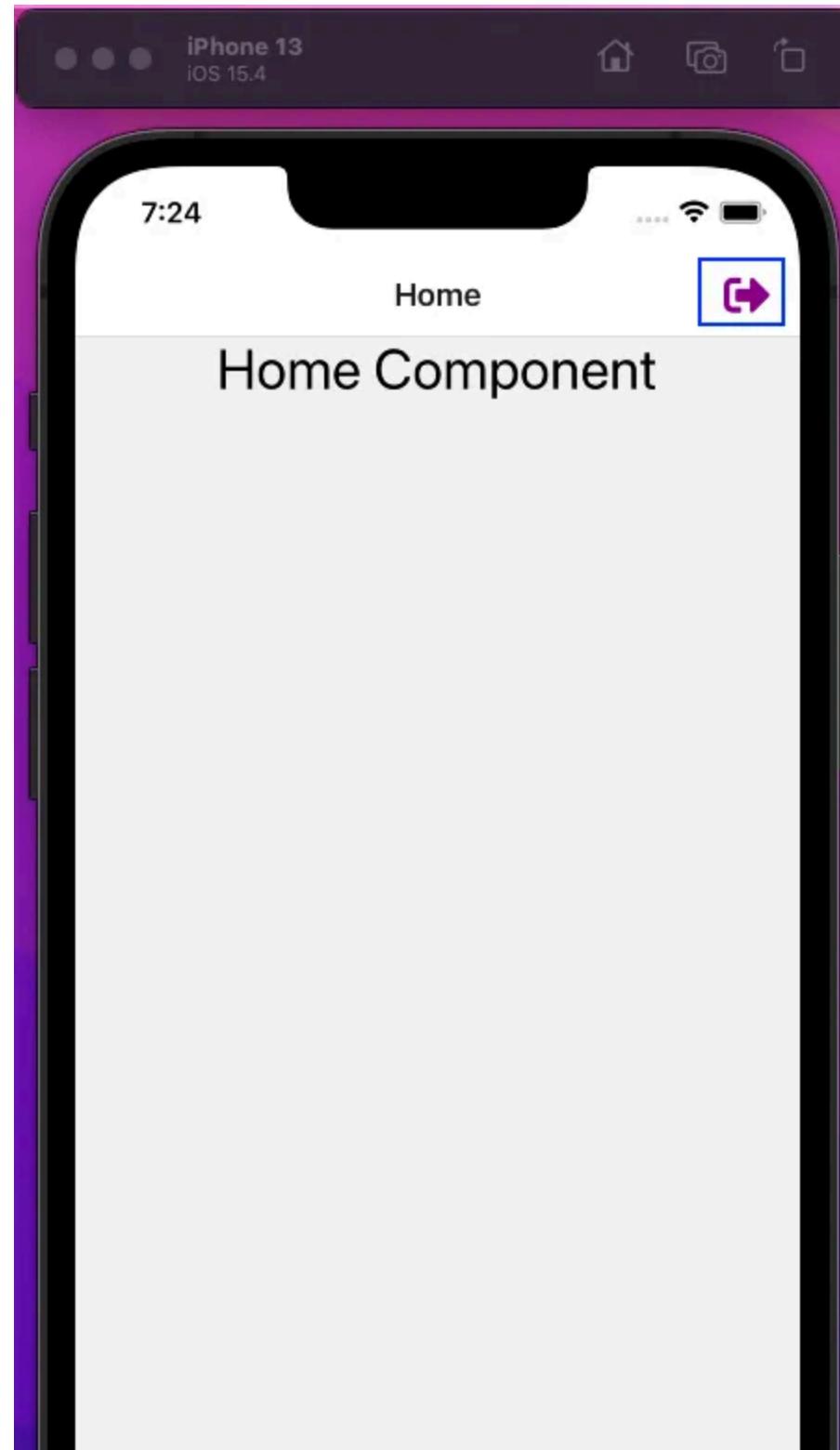
Now, create a **header** folder inside the **components** folder and a **HeaderTabs.js** file inside it.

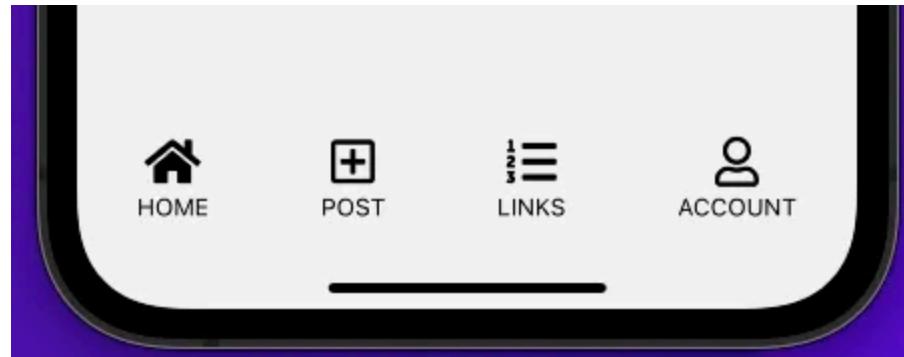
Here, we have a Signout button which will remove our Async Storage.

```
HeaderTabs.js - RNNODEMONGO
client > components > header > HeaderTabs.js > HeaderTabs > signOut
1 import { TouchableOpacity, SafeAreaView } from "react-native";
2 import React, { useContext } from 'react';
3 import { AuthContext } from "../../context/auth";
4 import FontAwesome5 from "react-native-vector-icons/FontAwesome5";
5 import AsyncStorage from "@react-native-async-storage/async-storage";
6
7 const HeaderTabs = () => {
8   const [state, setState] = useContext(AuthContext);
9
10  const signOut = async () => {
11    setState({ token: "", user: null });
12    await AsyncStorage.removeItem("auth-rn");
13  };
14
15  return (
16    <SafeAreaView>
17      <TouchableOpacity onPress={signOut}>
18        <FontAwesome5 name="sign-out-alt" size={25} color="darkmagenta" />
19      </TouchableOpacity>
20    </SafeAreaView>
21  );
22}
23
24 export default HeaderTabs
25
```

HeaderTabs.js

Now, our app have a working Signout button.



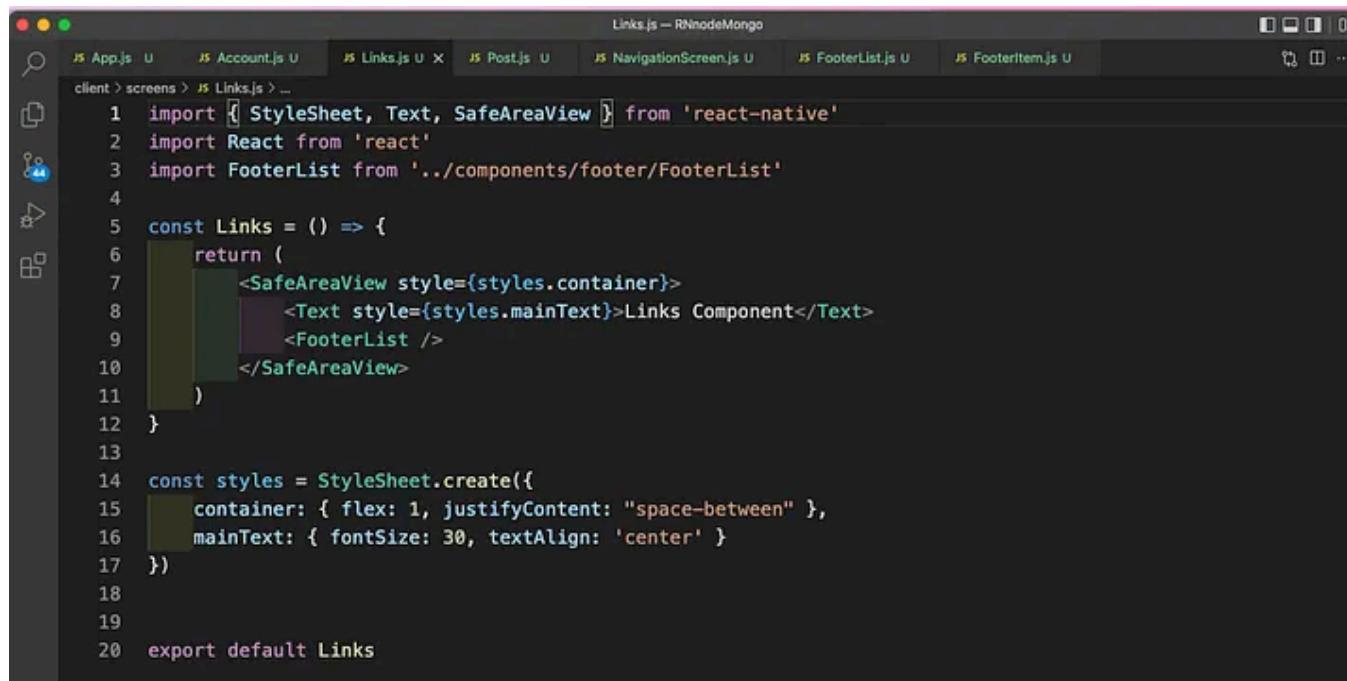


Now, we will create some new screens. So, first create a file **Account.js** inside the **screens** folder and add the below content in it.

```
Account.js - RNnodeMongo
client > screens > Account.js > ...
JS App.js U JS Account.js U X JS Links.js U JS Post.js U JS NavigationScreen.js U JS FooterList.js U JS FooterItem.js U ...
1 import { StyleSheet, Text, SafeAreaView } from 'react-native'
2 import React from 'react'
3 import FooterList from '../components/footer/FooterList'
4
5 const Account = () => {
6   return (
7     <SafeAreaView style={styles.container}>
8       <Text style={styles.mainText}>Account Component</Text>
9       <FooterList />
10    </SafeAreaView>
11  )
12}
13
14 const styles = StyleSheet.create({
15   container: { flex: 1, justifyContent: "space-between" },
16   mainText: { fontSize: 30, textAlign: 'center' }
17 })
18
19
20 export default Account
21
```

Account.js

Next create a file **Links.js** inside the **screens** folder and add the below content in it.



The screenshot shows a code editor window titled "Links.js - RNnodeMongo". The file path is "client > screens > Links.js". The code editor displays the following JavaScript code:

```
1 import { StyleSheet, Text, SafeAreaView } from 'react-native'
2 import React from 'react'
3 import FooterList from '../components/footer/FooterList'
4
5 const Links = () => {
6   return (
7     <SafeAreaView style={styles.container}>
8       <Text style={styles.mainText}>Links Component</Text>
9       <FooterList />
10    </SafeAreaView>
11  )
12}
13
14 const styles = StyleSheet.create({
15   container: { flex: 1, justifyContent: "space-between" },
16   mainText: { fontSize: 30, textAlign: 'center' }
17 })
18
19
20 export default Links
```

Links.js

Lastly, create a file **Post.js** inside the **screens** folder and add the below content in it.

```
Post.js - RNnodeMongo
JS App.js U JS Account.js U JS Links.js U JS Post.js U X JS NavigationScreen.js U JS FooterList.js U JS FooterItem.js U
client > screens > JS Post.js > ...
1 import { StyleSheet, Text, SafeAreaView } from 'react-native'
2 import React from 'react'
3 import FooterList from '../components/footer/FooterList'
4
5 const Post = () => {
6   return (
7     <SafeAreaView style={styles.container}>
8       <Text style={styles.mainText}>Post Component</Text>
9       <FooterList />
10    </SafeAreaView>
11  )
12}
13
14 const styles = StyleSheet.create({
15   container: { flex: 1, justifyContent: "space-between" },
16   mainText: { fontSize: 30, textAlign: 'center' }
17 })
18
19 export default Post
```

Post.js

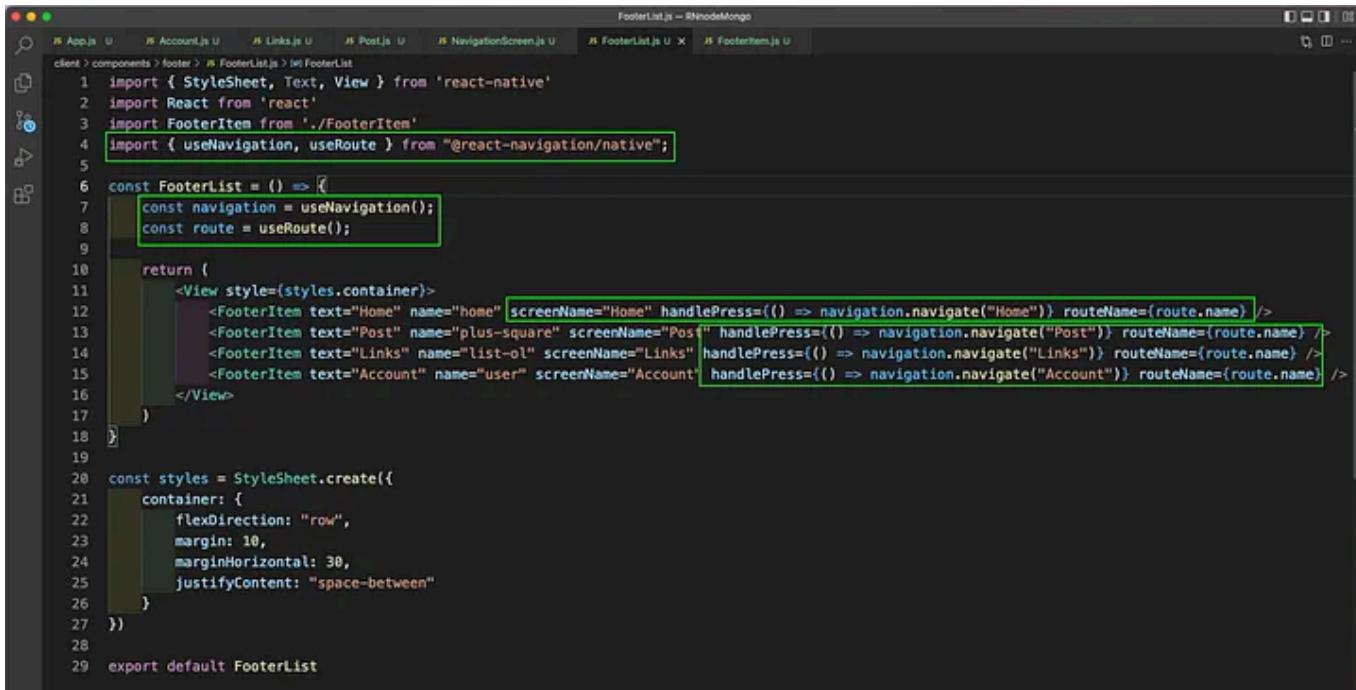
Now, we will add these new components route in our **NavigationScreen.js** file.

```
client > components > JS NavigationScreen.js > ↗ NavigationScreen
  1 import React, { useContext } from 'react'
  2 import { createNativeStackNavigator } from "@react-navigation/native-stack";
  3 import SignUp from '../screens/SignUp'
  4 import SignIn from '../screens/SignIn'
  5 import Home from '../screens/Home'
  6 import { AuthContext } from '../context/auth';
  7 import HeaderTabs from './header/HeaderTabs';
  8 import Account from '../screens/Account';
  9 import Post from '../screens/Post';
 10 import Links from '../screens/Links';
 11
 12 const Stack = createNativeStackNavigator();
 13
 14 const NavigationScreen = () => {
 15   const [state, setState] = useContext(AuthContext);
 16   const authenticated = state && state.token !== "" && state.user !== null;
 17
 18   return (
 19     <Stack.Navigator initialRouteName="Home">
 20       {authenticated ?
 21         (
 22           <>
 23             <Stack.Screen name="Home" component={Home} options={{ headerRight: () => <HeaderTabs />}}/>
 24             <Stack.Screen name="Account" component={Account} />
 25             <Stack.Screen name="Post" component={Post} />
 26             <Stack.Screen name="Links" component=[Links] />
 27           </>
 28         ) :
 29           <>
 30             <Stack.Screen name="SignIn" component={SignIn} />
 31             <Stack.Screen name="SignUp" component={SignUp} />
 32           </>
 33       )
 34     </Stack.Navigator>
 35   )
}

```

NavigationScreen.js

Next, in the **FooterList.js** file we will add three new props to the **FooterItem** component. They are `routeName`, `screenName` and `handlePress`.



The screenshot shows a code editor window with the file 'FooterList.js' open. The code is written in JavaScript and uses React Native components. It defines a 'FooterList' component that returns a horizontal list of four items: 'Home', 'Post', 'Links', and 'Account'. Each item is represented by a 'FooterItem' component. The 'FooterItem' component has properties for 'text', 'name', 'screenName', and 'handlePress'. The 'FooterList' component also includes a style sheet for its container.

```
client > components > footer > FooterList.js > FooterItem.js
1 import { StyleSheet, Text, View } from 'react-native'
2 import React from 'react'
3 import FooterItem from './FooterItem'
4 import { useNavigation, useRoute } from "@react-navigation/native";
5
6 const FooterList = () => {
7   const navigation = useNavigation();
8   const route = useRoute();
9
10  return (
11    <View style={styles.container}>
12      <FooterItem text="Home" name="home" screenName="Home" handlePress={() => navigation.navigate("Home")} routeName={route.name} />
13      <FooterItem text="Post" name="plus-square" screenName="Post" handlePress={() => navigation.navigate("Post")} routeName={route.name} />
14      <FooterItem text="Links" name="list-ol" screenName="Links" handlePress={() => navigation.navigate("Links")} routeName={route.name} />
15      <FooterItem text="Account" name="user" screenName="Account" handlePress={() => navigation.navigate("Account")} routeName={route.name} />
16    </View>
17  )
18}
19
20 const styles = StyleSheet.create({
21   container: {
22     flexDirection: "row",
23     margin: 10,
24     marginHorizontal: 30,
25     justifyContent: "space-between"
26   }
27 })
28
29 export default FooterList
```

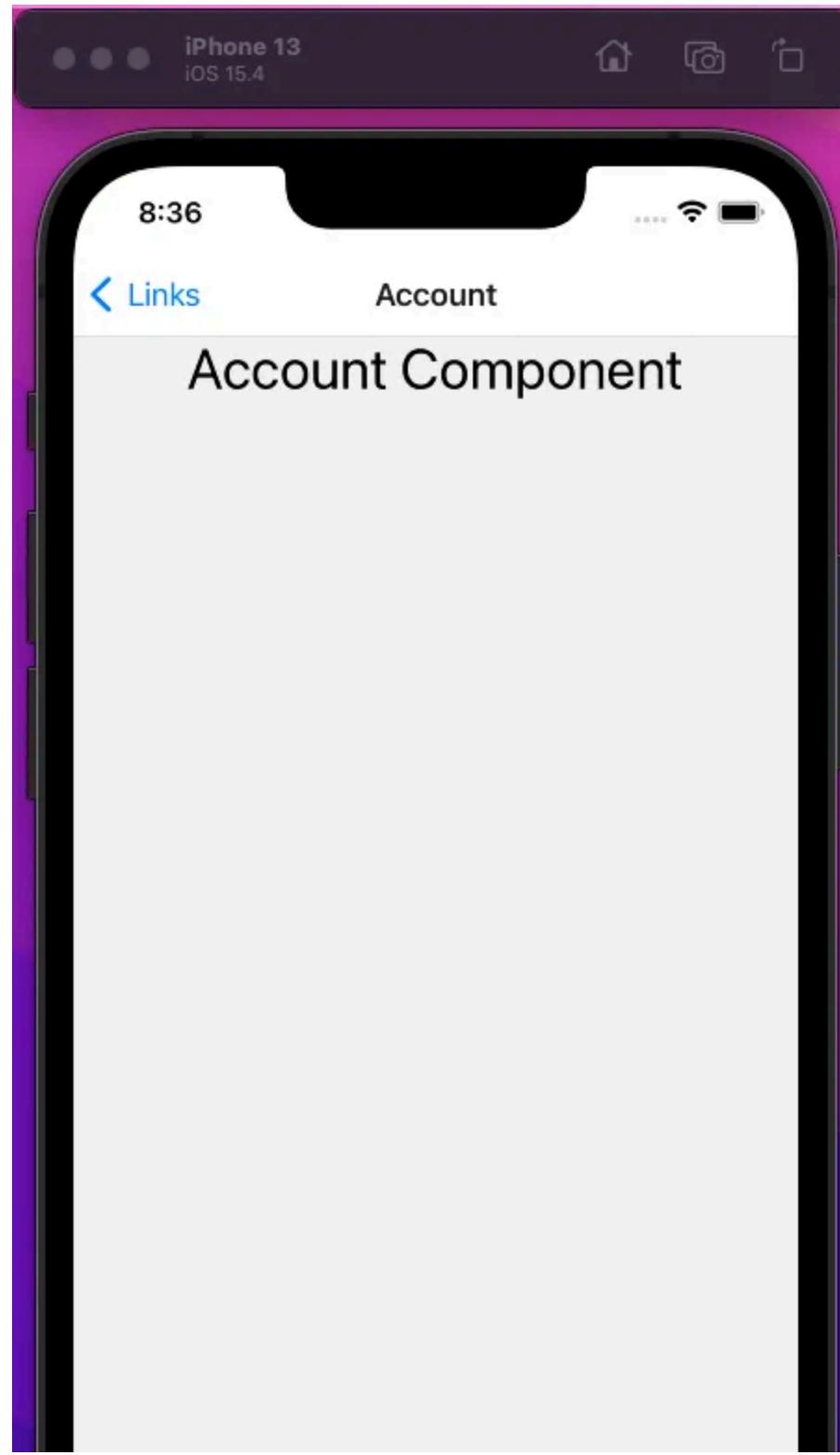
FooterList.js

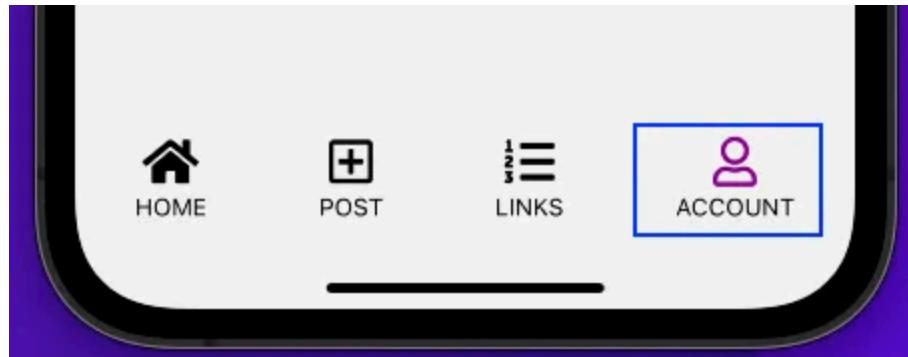
Now, we will update our **FooterItem.js** file. Here, by using the `screenName` and `routeName` we are showing an active tab. Also, `onPress` have the `handlePress` function.

```
JS App.js U JS Account.js U JS Links.js U JS Post.js U JS NavigationScreen.js U JS FooterList.js U JS FooterItem.js U X 08
client > components > footer > FooterItem.js > FooterItem
1 import { StyleSheet, Text, TouchableOpacity } from 'react-native'
2 import React from 'react'
3 import FontAwesome5 from "react-native-vector-icons/FontAwesome5";
4
5 const FooterItem = ({ name, text, handlePress, screenName, routeName }) => {
6   const activeScreenColor = screenName === routeName && "darkmagenta";
7
8   return (
9     <TouchableOpacity onPress={handlePress}>
10       <>
11         <FontAwesome5 name={name} size={25} style={styles.fontStyle} color={activeScreenColor} />
12         <Text style={styles.iconText}>{text}</Text>
13       </>
14     </TouchableOpacity>
15   )
16 }
17
18 const styles = StyleSheet.create({
19   fontStyle: { marginBottom: 3, alignSelf: "center" },
20   iconText: { fontSize: 12, textAlign: 'center', textTransform: 'uppercase' }
21 })
22
23 export default FooterItem
```

FooterItem.js

Now, our components are changing perfectly with active one shown in purple.



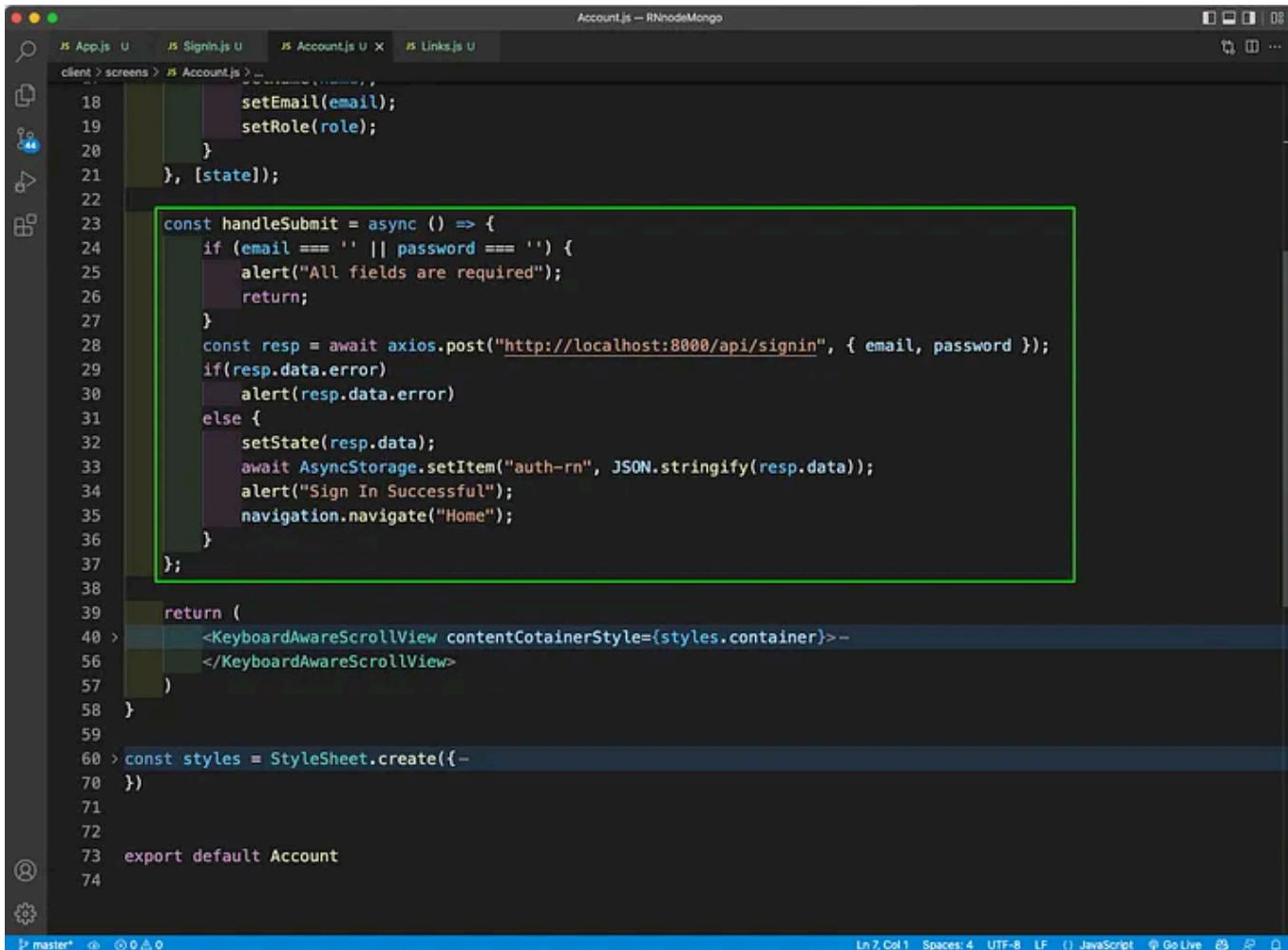


Now, we will update our `Account.js` file to first have the required imports and `useState`. We are also getting the name, email and role from the context API.

```
JS App.js U JS Signin.js U JS Account.js U JS Links.js U
client > screens > JS Account.js ...
1 import { StyleSheet, Text, View, TextInput, TouchableOpacity, Image } from 'react-native'
2 import { KeyboardAwareScrollView } from "react-native-keyboard-aware-scroll-view";
3 import axios from 'axios';
4 import AsyncStorage from "@react-native-async-storage/async-storage";
5 import React, { useContext, useEffect, useState } from 'react'
6 import { AuthContext } from '../context/auth';
7 const Account = ({ navigation }) => {
8   const [email, setEmail] = useState("");
9   const [name, setName] = useState("");
10  const [password, setPassword] = useState("");
11  const [role, setRole] = useState("");
12  const [state, setState] = useContext(AuthContext);
13  useEffect(() => {
14    if (state) {
15      const { name, email, role } = state.user;
16      setName(name);
17      setEmail(email);
18      setRole(role);
19    }
20  }, [state]);
21
22  const handleSubmit = async () => {
23  };
24
25  return (
26    <KeyboardAwareScrollView contentContainerStyle={styles.container}> ...
27    </KeyboardAwareScrollView>
28  );
29
30  const styles = StyleSheet.create({
31  });
32
33  export default Account
```

Account.js

Next, we will create the handleSubmit which is quite similar to the SignIn component.



```
client > screens > JS Account.js > ...
18     setEmail(email);
19     setRole(role);
20   }
21 }, [state]);
22
23 const handleSubmit = async () => {
24   if (email === '' || password === '') {
25     alert("All fields are required");
26     return;
27   }
28   const resp = await axios.post("http://localhost:8000/api/signin", { email, password });
29   if(resp.data.error)
30     alert(resp.data.error)
31   else {
32     setState(resp.data);
33     await AsyncStorage.setItem("auth-rn", JSON.stringify(resp.data));
34     alert("Sign In Successful");
35     navigation.navigate("Home");
36   }
37 };
38
39 return (
40   <KeyboardAwareScrollView contentContainerStyle={styles.container}>-
41   </KeyboardAwareScrollView>
42 )
43
44 > const styles = StyleSheet.create({
45   container: {
46     flex: 1,
47     padding: 20
48   }
49 }
50
51
52
53 export default Account
54
```

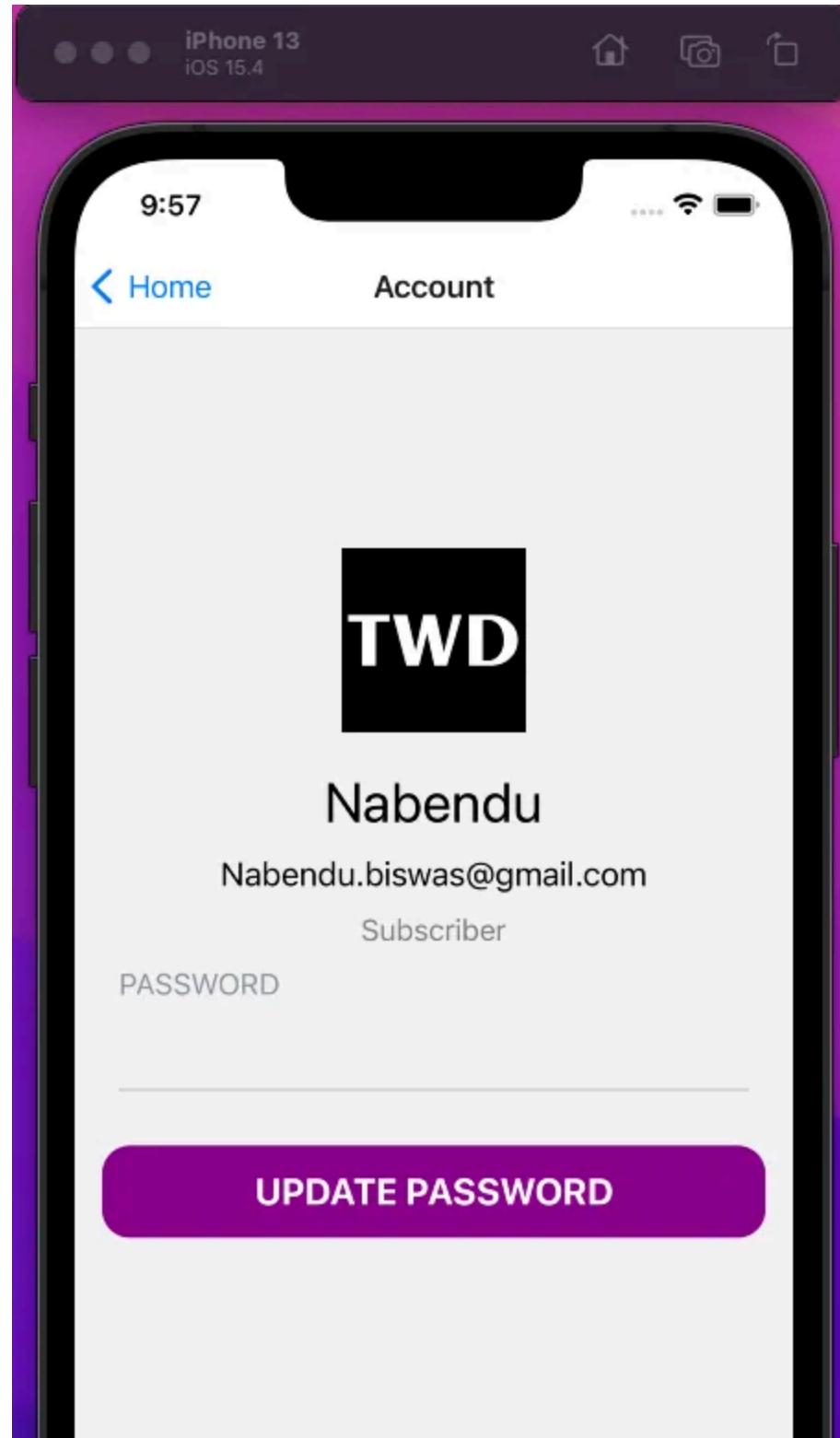
Account.js

Lastly we will add the code in return and the styles for it. Again it is quite similar to the SignIn component.

```
Account.js - iNodeMongo
client > screens > Account.js > [M] Account > [M] handleSubmit
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
return (
  <KeyboardAwareScrollView contentContainerStyle={styles.container}>
    <View style={{ marginVertical: 100 }}>
      <View style={styles.imageContainer}>
        <Image source={require("../assets/logo.png")} style={styles.imageStyles} />
      </View>
      <Text style={styles.signupText}>{name}</Text>
      <Text style={styles.emailText}>{email}</Text>
      <Text style={styles.roleText}>{role}</Text>
      <View style={{ marginHorizontal: 24 }}>
        <Text style={{ fontSize: 16, color: '#8e93a1' }}>PASSWORD</Text>
        <TextInput style={styles.signupInput} value={password} onChangeText={(text => setPassword(text))} secureTextEntry={true} autoCompleteType="password" />
      </View>
      <TouchableOpacity onPress={handleSubmit} style={styles.buttonStyle}>
        <Text style={styles.buttonText}>Update Password</Text>
      </TouchableOpacity>
    </View>
  </KeyboardAwareScrollView>
)
const styles = StyleSheet.create({
  container: { flex: 1, justifyContent: 'center' },
  signupText: { fontSize: 30, textAlign: 'center', paddingBottom: 10 },
  emailText: { fontSize: 18, textAlign: 'center', paddingBottom: 10 },
  roleText: { fontSize: 16, textAlign: 'center', paddingBottom: 10, color: 'gray' },
  signupInput: { borderBottomWidth: 0.5, height: 48, borderBottomColor: "#8e93a1", marginBottom: 30 },
  buttonStyle: { backgroundColor: "darkmagenta", height: 50, justifyContent: "center", marginHorizontal: 15, borderRadius: 15 },
  buttonText: { fontSize: 20, textAlign: 'center', color: '#fff', textTransform: 'uppercase', fontWeight: 'bold' },
  imageContainer: { justifyContent: "center", alignItems: "center" },
  imageStyles: { width: 100, height: 100, marginVertical: 20 }
})
```

Account.js

Now, our App will show this nice view in Account.





Now, we will add the logic to upload image in Account.js file. Here, we are creating a image state variable and having a hard-coded url as of now.

The screenshot shows a code editor window with the title "Account.js -- RNnodeMongo". The tab bar includes "JS App.js U", "JS Signin.js U", "JS Account.js U X", and "JS Links.js U". The code editor displays the following JavaScript code:

```
1 import { StyleSheet, Text, View, TextInput, TouchableOpacity, Image } from 'react-native'
2 import { KeyboardAwareScrollView } from "react-native-keyboard-aware-scroll-view";
3 import axios from 'axios';
4 import AsyncStorage from "@react-native-async-storage/async-storage";
5 import React, { useContext, useEffect, useState } from 'react'
6 import { AuthContext } from '../context/auth';
7 import FontAwesome5 from "react-native-vector-icons/FontAwesome5";
8
9 const Account = ({ navigation }) => {
10   const [email, setEmail] = useState("");
11   const [name, setName] = useState("");
12   const [password, setPassword] = useState("");
13   const [role, setRole] = useState("");
14   const [image, setImage] = useState({ url: "https://cdn.pixabay.com/photo/2015/01/08/18/25/
desk-593327_960_720.jpg", public_id: "" });
15   const [state, setState] = useContext(AuthContext);
16
17   useEffect(() => {
18     if (state) {
19       const { name, email, role, image } = state.user;
20       setName(name);
21       setEmail(email);
22       setRole(role);
23     }
24   }, [state]);
25
26 >   const handleSubmit = async () => {...}
27   };
28   const handleUpload = () => {};
29
30   return [
31     <KeyboardAwareScrollView contentContainerStyle={styles.container}>
32       <View style={{ marginVertical: 100 }}>
33         <View style={styles.imageContainer}>
34           {image && image.url ? <Image source={{ uri: image.url }} style={styles.imageStyles} /> : (
35             <Image source={require('../assets/camera.png')} style={styles.imageStyles} />
36           )
37         </View>
38       </View>
39     </KeyboardAwareScrollView>
40   ];
41 }
```

The code uses React Native components like `KeyboardAwareScrollView`, `Text`, `View`, `TextInput`, `TouchableOpacity`, and `Image`. It also utilizes `useState` and `useEffect` hooks. The `AuthContext` is used to get user information. A placeholder image URL is provided for the `image` state.

Account.js

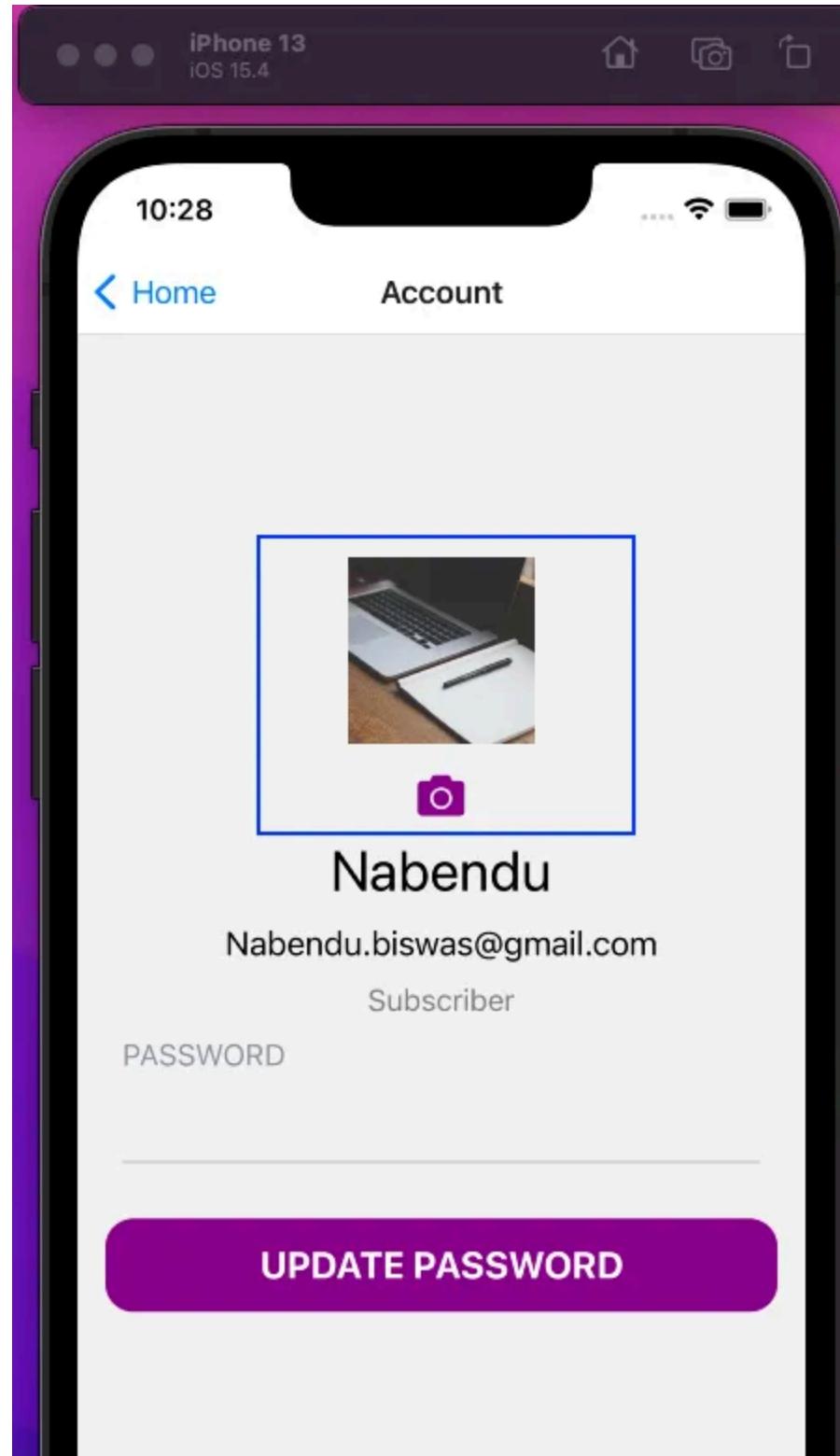
Next, we will show an image and also a camera icon to upload image. Notice that when we will not have a image for the first time, we are showing the camera icon.

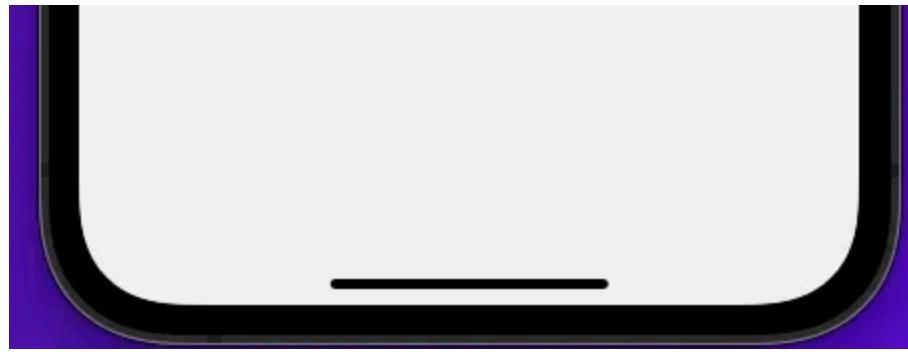
```
Account.js - RNnodeMongo
client > screens > Account.js > Account
41 const handleUpload = () => {};
42
43 return [
44   <KeyboardAwareScrollView contentContainerStyle={styles.container}>
45     <View style={{ marginVertical: 100 }}>
46       <View style={styles.imageContainer}>
47         {image && image.url ? <Image source={{ uri: image.url }} style={styles.imageStyles} /> : (
48           <TouchableOpacity onPress={() => handleUpload()}>
49             <FontAwesome5 name="camera" size={25} color="darkmagenta" />
50           </TouchableOpacity>
51         )}
52       </View>
53       {image && image.url ? (
54         <TouchableOpacity onPress={() => handleUpload()}>
55           <FontAwesome5 name="camera" size={25} color="darkmagenta" style={styles.iconStyle} />
56         </TouchableOpacity>
57       ) : (
58         <></>
59       )}
60       <Text style={styles.signupText}>{name}</Text>
61       <Text style={styles.emailText}>{email}</Text>
62       <Text style={styles.roleText}>{role}</Text>
63     <View style={{ marginHorizontal: 24 }}>-
64     </View>
65     <TouchableOpacity onPress={handleSubmit} style={styles.buttonStyle}>-
66     </TouchableOpacity>
67   </View>
68 </KeyboardAwareScrollView>
69
70 }
71
72 }
73
74
75 const styles = StyleSheet.create({
76   iconStyle: { marginTop: -5, marginBottom: 10, alignSelf: "center" },
77   container: { flex: 1, justifyContent: 'center' },
78   signupText: { fontSize: 30, textAlign: 'center', paddingBottom: 10 },
79 })

```

Account.js

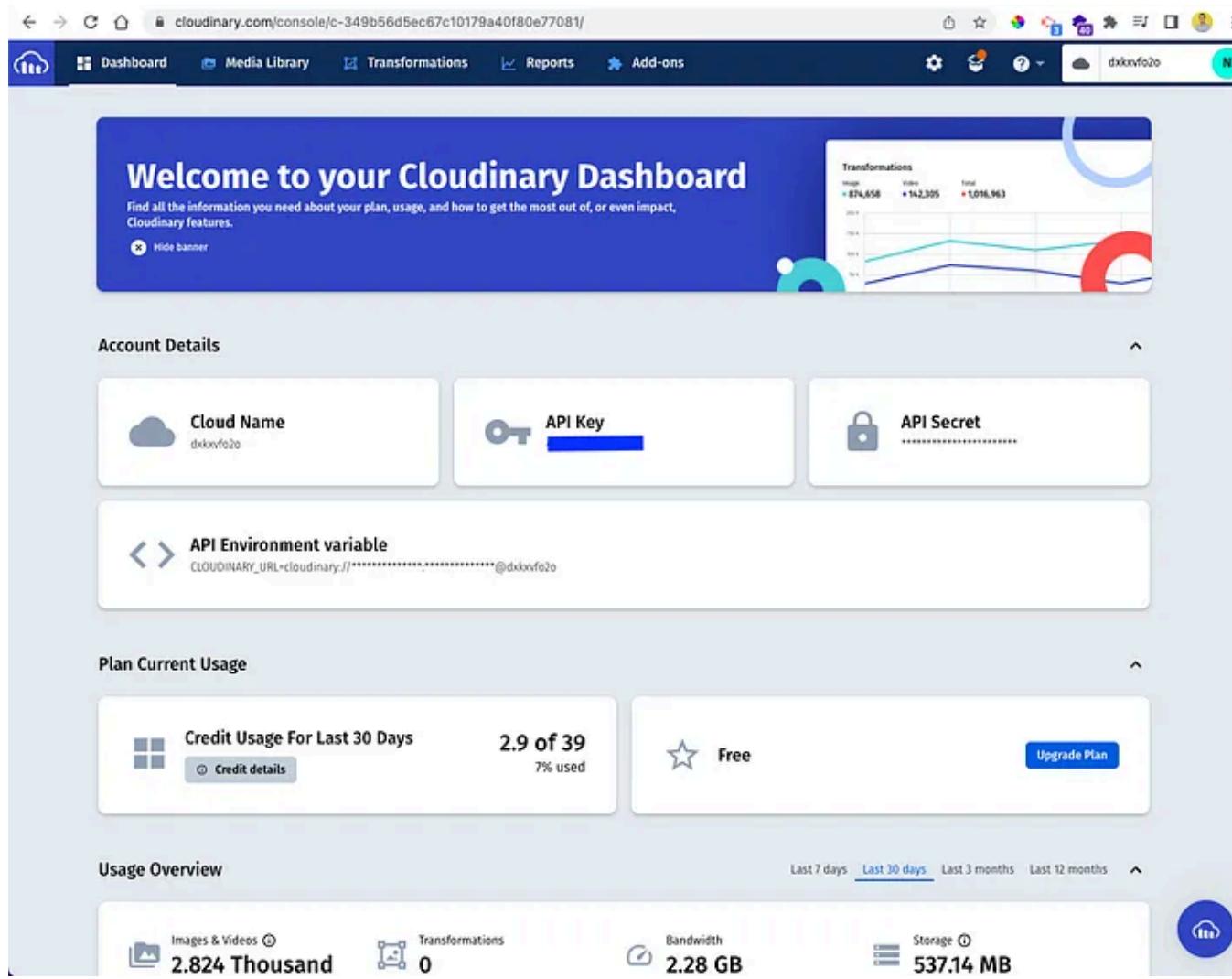
Now, a nice image is shown in the app.





Now, we will use cloudinary to upload our images. So, login to your cloudinary account.

[cloudinary.com/console/c-349b56d5ec67c10179a40f80e77081/](https://cloudinary.com/console/c-349b56d5ec67c10179a40f80e77081/)



The screenshot shows the Cloudinary Dashboard interface. At the top, there's a banner with the text "Welcome to your Cloudinary Dashboard" and a "Hide banner" button. To the right of the banner is a "Transformations" chart showing usage statistics: Image (874,658), Video (142,395), and Total (1,016,963). Below the banner are sections for "Account Details" (Cloud Name: dxkxvfo2o, API Key, API Secret), "API Environment variable" (CLOUDINARY\_URL: clouddinary://\*\*\*\*\*@dxkxvfo2o), and "Plan Current Usage" (Credit Usage For Last 30 Days: 2.9 of 39, Plan: Free, Upgrade Plan button). The "Usage Overview" section at the bottom provides metrics for Images & Videos (2.824 Thousand), Transformations (0), Bandwidth (2.28 GB), and Storage (537.14 MB). A blue "Upload" button is located on the far right.

Welcome to your Cloudinary Dashboard

Find all the information you need about your plan, usage, and how to get the most out of, or even impact, Cloudinary features.

Hide banner

Transformations

Image 874,658 Video 142,395 Total 1,016,963

Cloud Name dxkxvfo2o

API Key

API Secret

API Environment variable

CLOUDINARY\_URL: clouddinary://\*\*\*\*\*@dxkxvfo2o

Credit Usage For Last 30 Days 2.9 of 39 7% used

Credit details

Free

Upgrade Plan

Last 7 days Last 30 days Last 3 months Last 12 months

Images & Videos 2.824 Thousand

Transformations 0

Bandwidth 2.28 GB

Storage 537.14 MB

Upload

After that click on Settings and then Upload tab.

The screenshot shows the Cloudinary settings interface. The top navigation bar includes links for Dashboard, Media Library, Transformations, Reports, and Add-ons. The 'Settings' section is active, with tabs for Account, Upload (which is selected and highlighted with a blue border), Security, and Users. On the right side, there's a sidebar titled 'Self-Service Operations' containing links for Bulk delete and Upload directly from my own bucket. Another sidebar titled 'Structured Metadata' has a link to Manage Structured Metadata. The main content area on the left contains sections for Automatic backup (disabled), Use your own backup bucket (disabled), Auto-create folders (enabled), and Auto upload mapping. The Auto upload mapping section includes fields for Folder and URL prefix, and a link to Add another mapping. A note explains that this feature maps a Cloudinary folder to an external URL for auto-uploading. Below this, there's an example: 'For example: myfolder -> s3://my-bucket/my-path/'. The top right corner shows the user's plan as 'Free Plan' with 39 Credits (2.9 (7%)).

After scrolling a bit down, we will get Upload presets. Here the ml\_default should be signed.

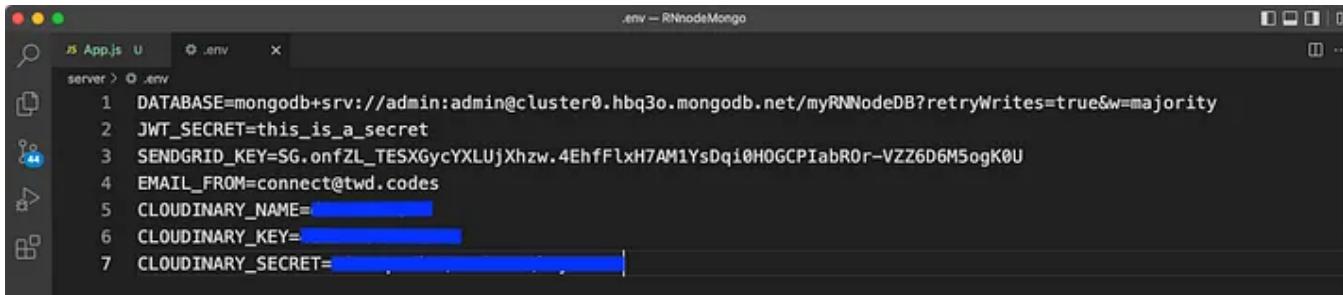
The screenshot shows the Cloudinary console interface for managing upload presets. At the top, there's a navigation bar with links for Dashboard, Media Library, Transformations, Reports, and Add-ons. On the right side of the header, there's a user profile icon and a search bar.

The main area displays a table of upload presets:

Name	Mode	Settings
react_cloudinary	Signed	Overwrite: true Unique filename: true Delivery type: upload Access mode: public Folder: react_cloudinary
cloudinary_react	Signed	Overwrite: true Unique filename: true Delivery type: upload Access mode: public Folder: cloudinary_react
BillingRestro	Unsigned	Unique filename: true Delivery type: upload Access mode: public Incoming Transformation: c_scale,q_auto,w_180 Eager Transformations: c_scale,h_1000,q_auto,w_1000 Folder: BillingRestro
sickfits	Unsigned	Unique filename: true Delivery type: upload Access mode: public Incoming Transformation: c_scale,q_auto,w_500 Eager Transformations: c_scale,q_auto,w_1000 Folder: sickfits
ml_default	Signed	Overwrite: true Use filename or externally defined Public ID: true Unique filename: true

At the bottom left, there's a button labeled "Add upload preset" and a descriptive text about what upload presets are used for.

Now, we will take the cloudinary name, API key and secrets from the home page and create variables from the same in .env file inside server.



.env — RNnodeMongo

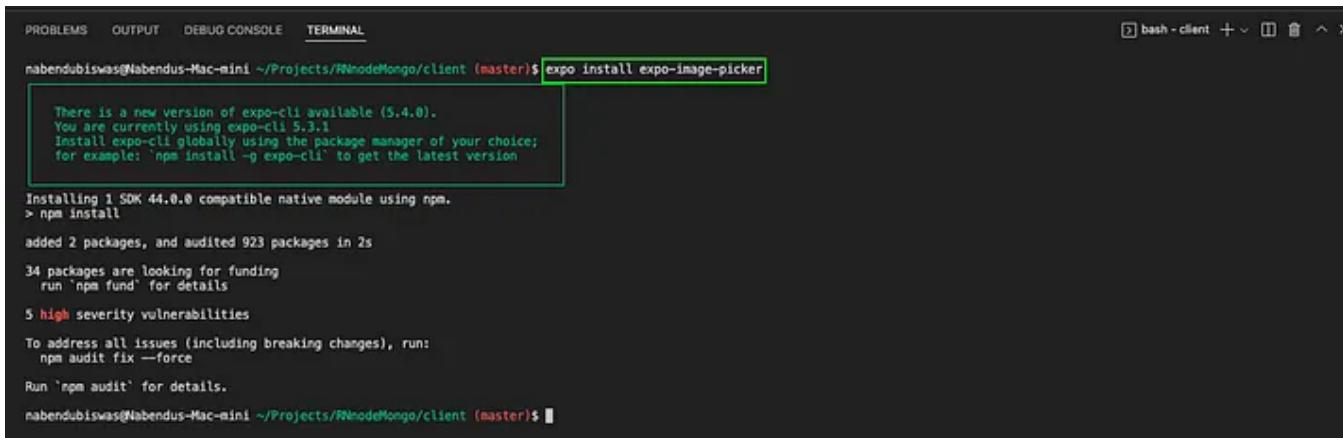
JS App.js U ⌂ .env X

server > ⌂ .env

```
1 DATABASE=mongodb+srv://admin:admin@cluster0.hbq3o.mongodb.net/myRNNodeDB?retryWrites=true&w=majority
2 JWT_SECRET=this_is_a_secret
3 SENDGRID_KEY=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
4 EMAIL_FROM=connect@twd.codes
5 CLOUDINARY_NAME=[REDACTED]
6 CLOUDINARY_KEY=[REDACTED]
7 CLOUDINARY_SECRET=[REDACTED]
```

.env

Now, we will add expo image picker in our client side.



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

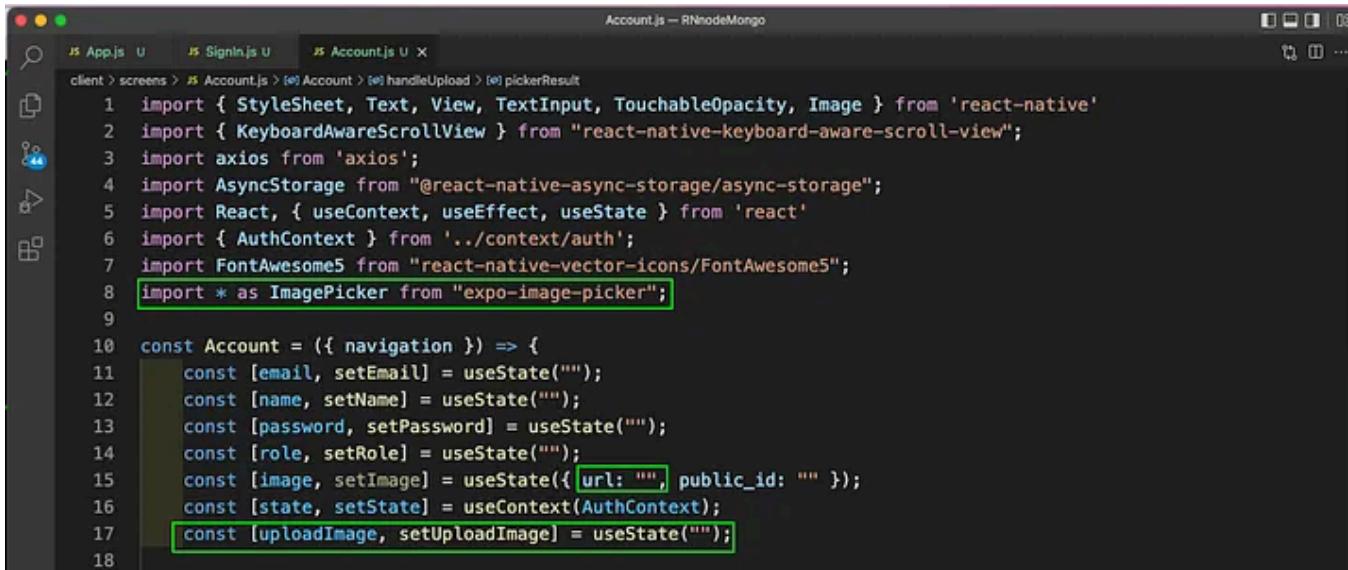
nabendubiswas@Nabendus-Mac-mini ~/Projects/RNnodeMongo/client (master)\$ expo install expo-image-picker

There is a new version of expo-cli available (5.4.0).  
You are currently using expo-cli 5.3.1  
Install expo-cli globally using the package manager of your choice;  
for example: 'npm install -g expo-cli' to get the latest version

Installing 1 SDK 44.0.0 compatible native module using npm.  
> npm install  
added 2 packages, and audited 923 packages in 2s  
34 packages are looking for funding  
 run 'npm fund' for details  
5 **high** severity vulnerabilities  
To address all issues (including breaking changes), run:  
 npm audit fix --force  
Run 'npm audit' for details.

nabendubiswas@Nabendus-Mac-mini ~/Projects/RNnodeMongo/client (master)\$

Next, we will use it in the **Account.js** file. Here, we are adding a new state variable and also removed the hard-coded url.



A screenshot of a code editor window titled "Account.js — RNnodeMongo". The file path is "client > screens > Account.js". The code imports various React Native components and libraries. Several lines of code are highlighted with green boxes:

```
1 import { StyleSheet, Text, View, TextInput, TouchableOpacity, Image } from 'react-native'
2 import { KeyboardAwareScrollView } from "react-native-keyboard-aware-scroll-view";
3 import axios from 'axios';
4 import AsyncStorage from "@react-native-async-storage/async-storage";
5 import React, { useContext, useEffect, useState } from 'react'
6 import { AuthContext } from '../context/auth';
7 import FontAwesome5 from "react-native-vector-icons/FontAwesome5";
8 import * as ImagePicker from "expo-image-picker";
9
10 const Account = ({ navigation }) => {
11   const [email, setEmail] = useState("");
12   const [name, setName] = useState("");
13   const [password, setPassword] = useState("");
14   const [role, setRole] = useState("");
15   const [image, setImage] = useState({ url: "", public_id: "" });
16   const [state, setState] = useContext(AuthContext);
17   const [uploadImage, setUploadImage] = useState("");
18 }
```

Account.js

Now, we will update our handleUpload function. Here, we are using the ImagePicker functions to get the image and set the base64Image to uploadImage.

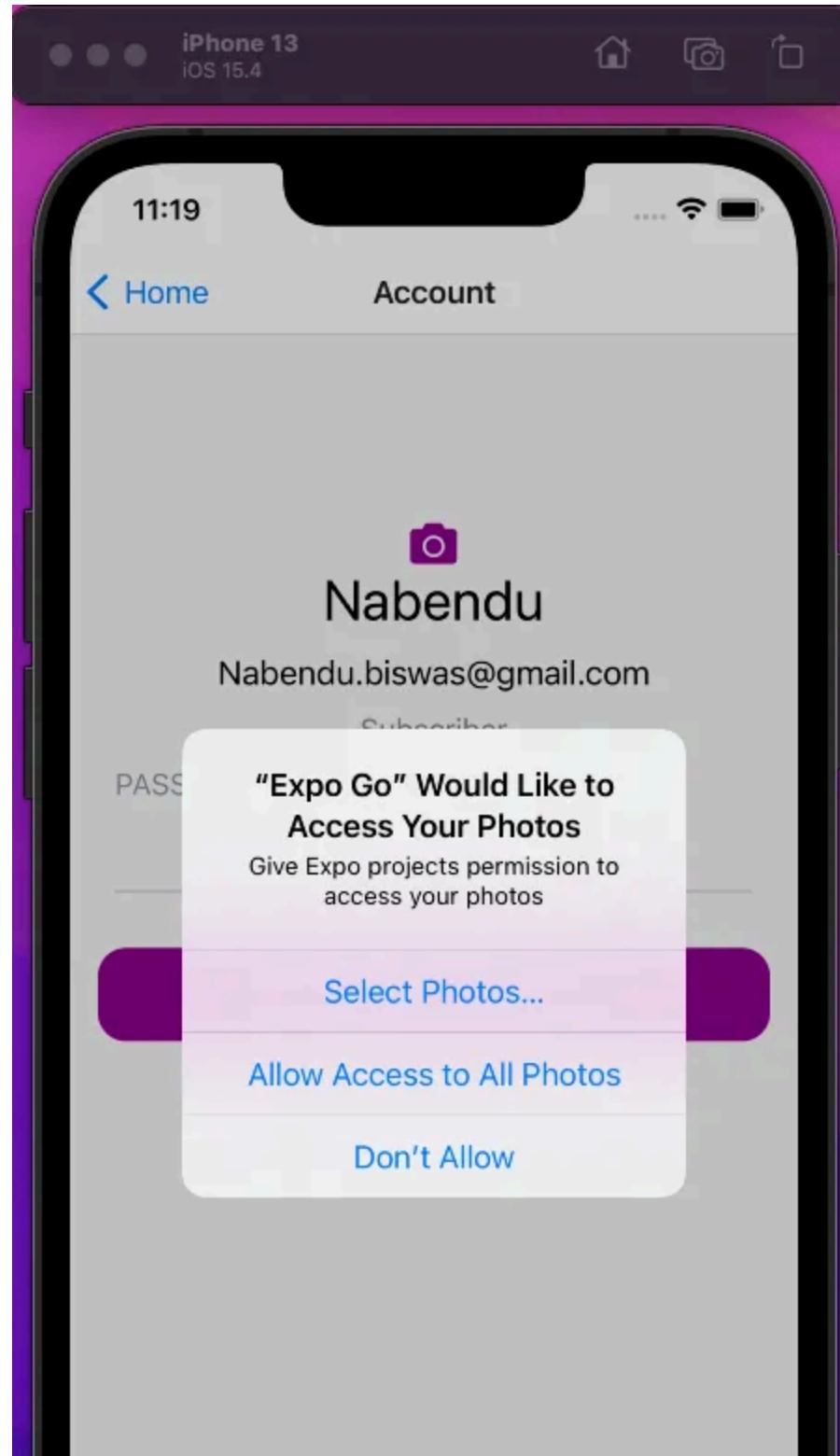
We are also passing this to the backend route, which we need to create.

```
Account.js -- RNnodeMongo
JS App.js U JS Signin.js U JS Account.js U X
Client > screens > JS Account.js > [8] Account > [6] handleUpload > [8] pickerResult
42    };
43    const handleUpload = async () => {
44      let permissionResult = await ImagePicker.requestMediaLibraryPermissionsAsync();
45      if (permissionResult.granted === false) {
46        alert("Camera access is required");
47        return;
48      }
49      let pickerResult = await ImagePicker.launchImageLibraryAsync({
50        allowsEditing: true,
51        aspect: [4, 3],
52        base64: true,
53      });
54      if (pickerResult.cancelled === true) {
55        return;
56      }
57      let base64Image = `data:image/jpg;base64,${pickerResult.base64}`;
58      setUploadImage(base64Image);
59      const { data } = await axios.post("http://localhost:8000/api/upload-image", {
60        image: base64Image,
61      });
62      console.log("UPLOADED RESPONSE => ", data);
63    };
64
65    return (
66      <KeyboardAwareScrollView contentContainerStyle={styles.container}>
67        <View style={{ marginVertical: 100 }}>
68          <View style={styles.imageContainer}>
69            {image && image.url ? <Image source={{ uri: image.url }} style={styles.imageStyles} /> :
70              uploadImage ? <Image source={{ uri: uploadImage }} style={styles.imageStyles} /> : (
71                <TouchableOpacity onPress={() => handleUpload()}>
72                  <FontAwesome5 name="camera" size={25} color="darkmagenta" />
73                
74              )
75            </View>
76            {image && image.url ? (

```

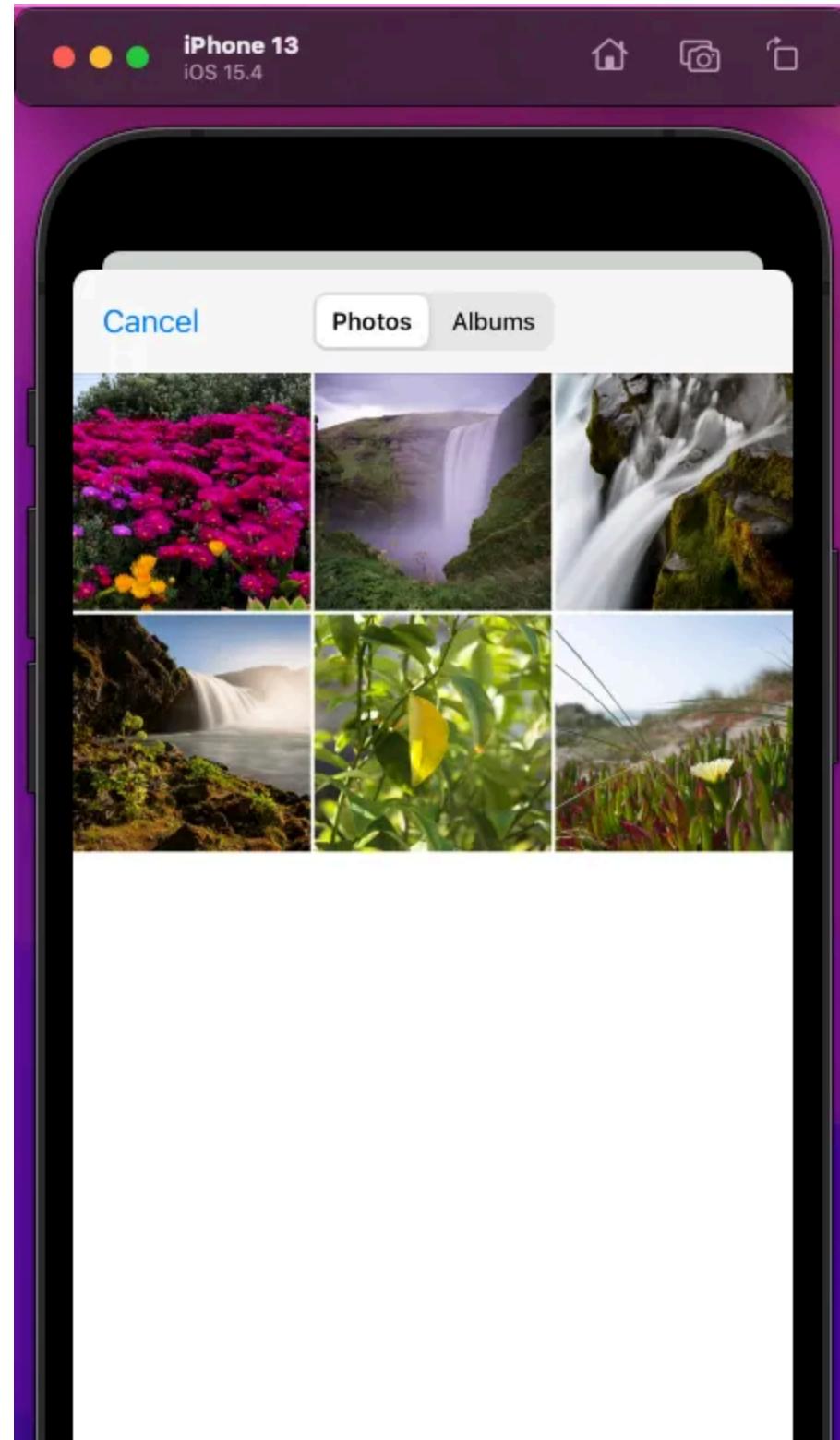
Account.js

Now, in the app when we click on the Upload button, we will first be asked for permissions.



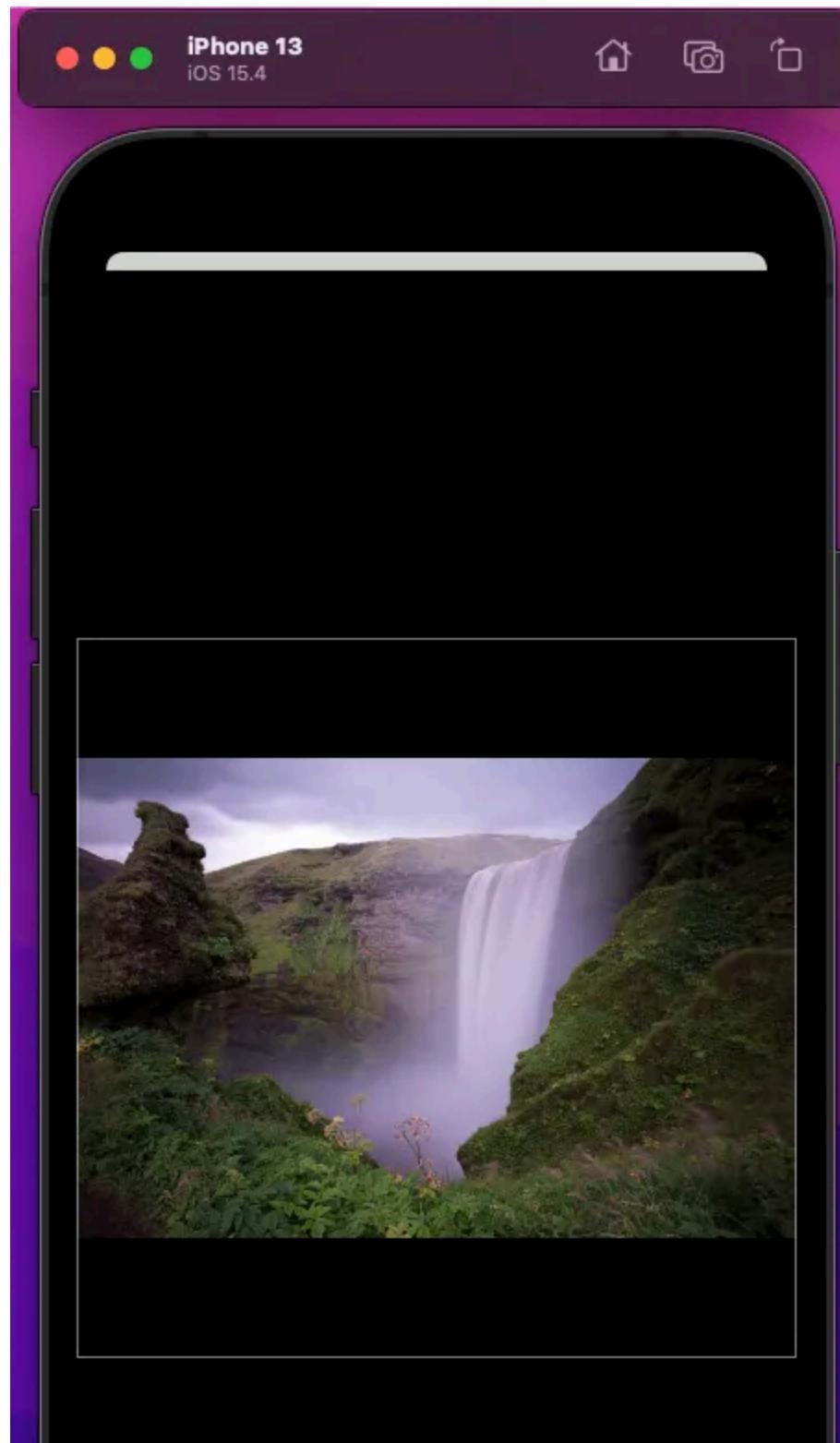


Next, we will be taken to the simulator photo gallery.



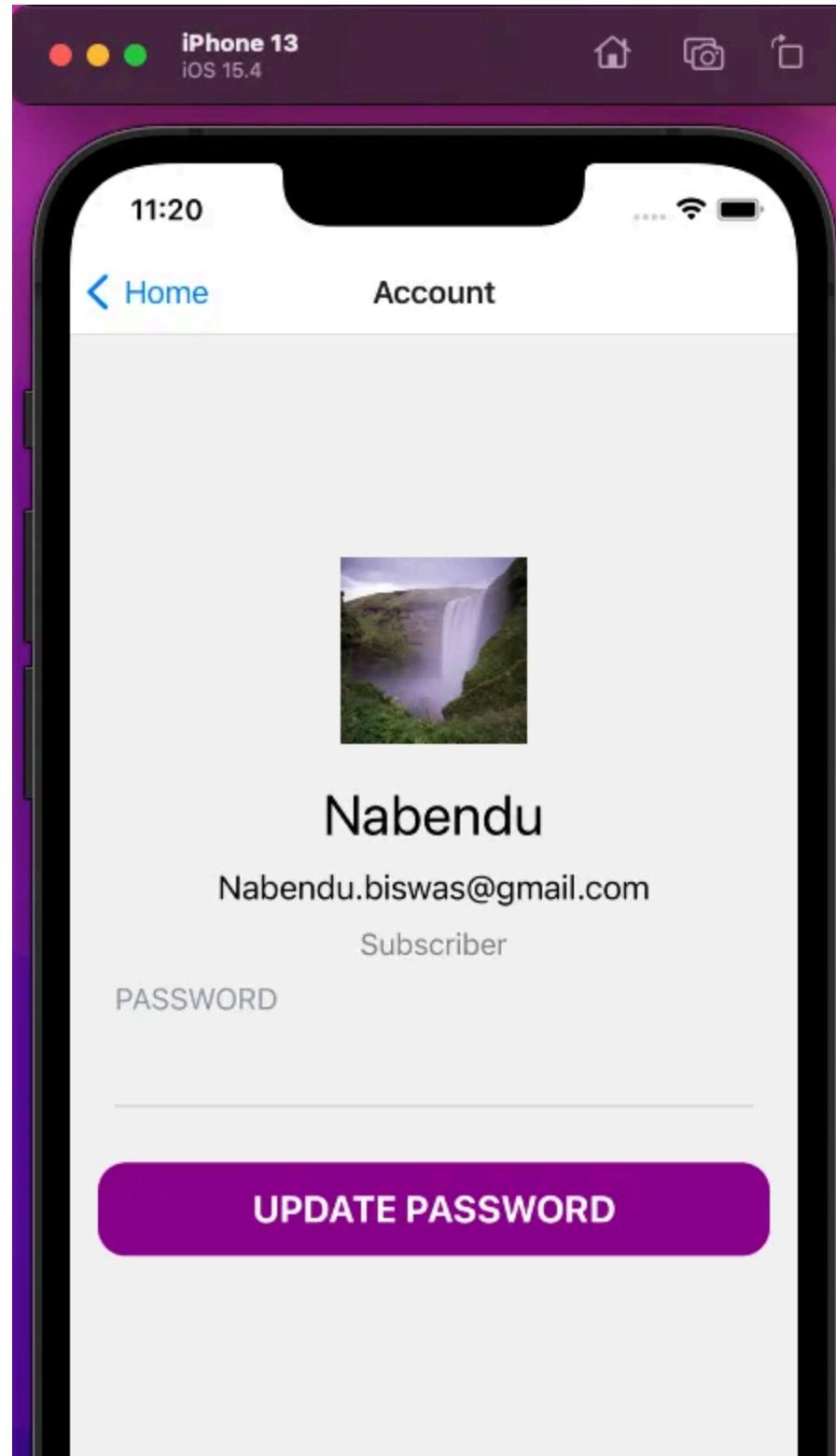


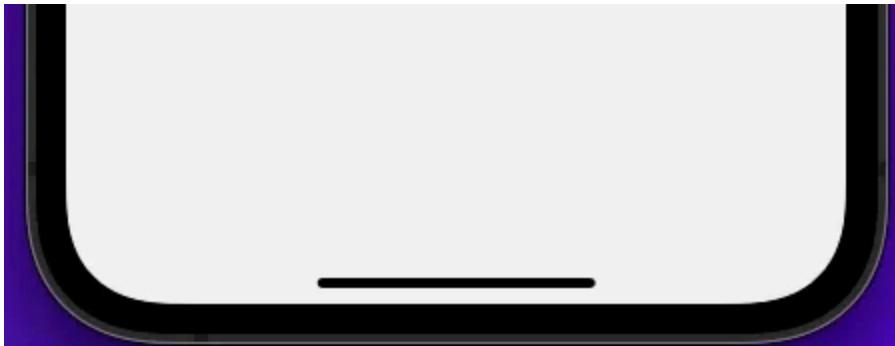
Here, on picking a image we will be taken to the next scree. Here, press on Choose.





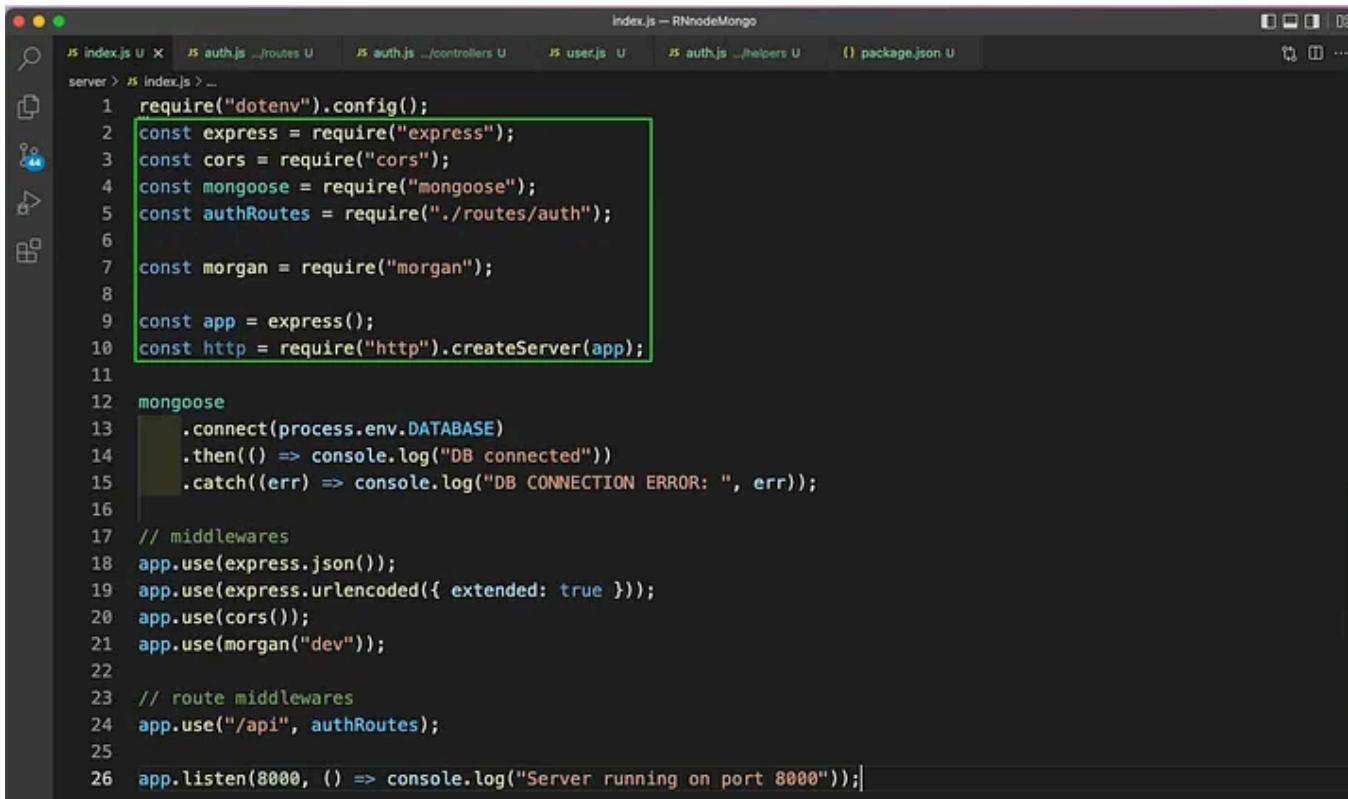
Now, our uploaded image will be shown in the component.





Next, we need to create the upload-image route in the server. But first we will change all of our imports to the old NodeJS kind import, because some old node\_modules were creating problem.

First update them in **index.js** file.



```
1 require("dotenv").config();
2 const express = require("express");
3 const cors = require("cors");
4 const mongoose = require("mongoose");
5 const authRoutes = require("./routes/auth");
6
7 const morgan = require("morgan");
8
9 const app = express();
10 const http = require("http").createServer(app);
11
12 mongoose
13   .connect(process.env.DATABASE)
14   .then(() => console.log("DB connected"))
15   .catch((err) => console.log("DB CONNECTION ERROR: ", err));
16
17 // middlewares
18 app.use(express.json());
19 app.use(express.urlencoded({ extended: true }));
20 app.use(cors());
21 app.use(morgan("dev"));
22
23 // route middlewares
24 app.use("/api", authRoutes);
25
26 app.listen(8000, () => console.log("Server running on port 8000"));
```

index.js

Next, we will update them in **auth.js** file in **routes** folder. Here, we are also creating a new route for **upload-image**.

```
auth.js — RNnodeMongo
JS index.js U JS auth.js U X JS Account.js U JS SignIn.js U JS SignUp.js U
Server > routes > JS auth.js > ...
1 const express = require("express");
2 const router = express.Router();
3
4 // controllers
5 const { signup, signin, forgotPassword, resetPassword, uploadImage } = require("../controllers/auth");
6
7 router.get("/", (req, res) => {
8   return res.json({
9     data: "hello world from the API",
10   });
11 });
12
13 router.post("/signup", signup);
14 router.post("/signin", signin);
15 router.post("/forgot-password", forgotPassword);
16 router.post("/reset-password", resetPassword);
17 router.post("/upload-image", uploadImage);
18
19 module.exports = router;
```

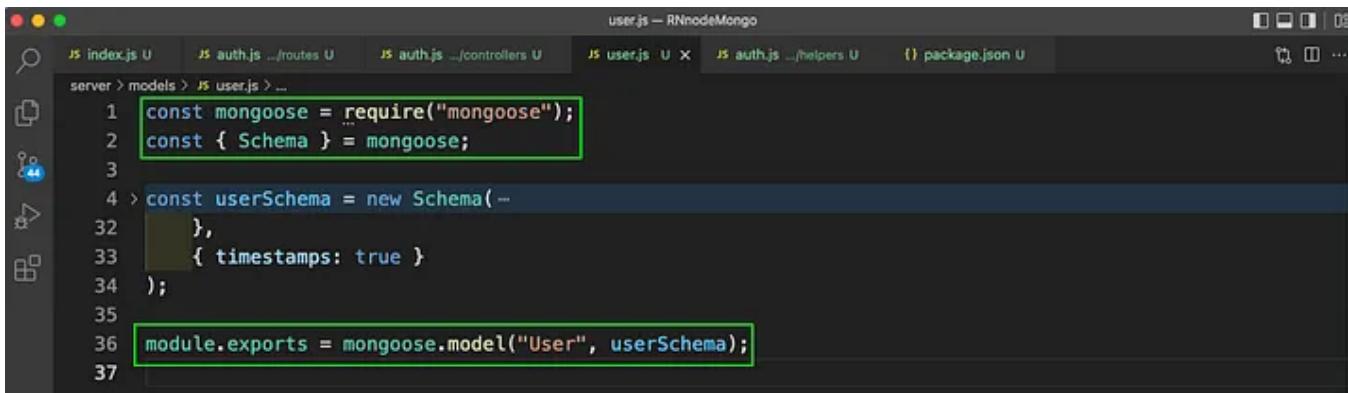
auth.js

Next, we will update the imports in **auth.js** for the **controllers**. We are also adding the cloudinary config.

```
auth.js — RNnodeMongo
JS index.js U JS auth.js U X JS Account.js U JS Signin.js U JS SignUp.js U
server > controllers > JS auth.js > ...
1 const User = require("../models/user");
2 const jwt = require("jsonwebtoken");
3 const { hashPassword, comparePassword } = require("../helpers/auth");
4 const { nanoid } = require("nanoid");
5 const cloudinary = require("cloudinary");
6
7 // cloudinary
8 cloudinary.config({
9   cloud_name: process.env.CLOUDINARY_NAME,
10  api_key: process.env.CLOUDINARY_KEY,
11  api_secret: process.env.CLOUDINARY_SECRET,
12 });
13
14 // sendgrid
15 require("dotenv").config();
16 const sgMail = require("@sendgrid/mail");
17 sgMail.setApiKey(process.env.SENDGRID_KEY);
18
19
20 > exports.signup = async (req, res) => {
21 };
22
23 > exports.signin = async (req, res) => {
24 };
25
26 > exports.forgotPassword = async (req, res) => {
27 };
28
29 > exports.resetPassword = async (req, res) => {
30 };
31
32 > exports.uploadImage = async (req, res) => {
33 };
34
35
```

auth.js

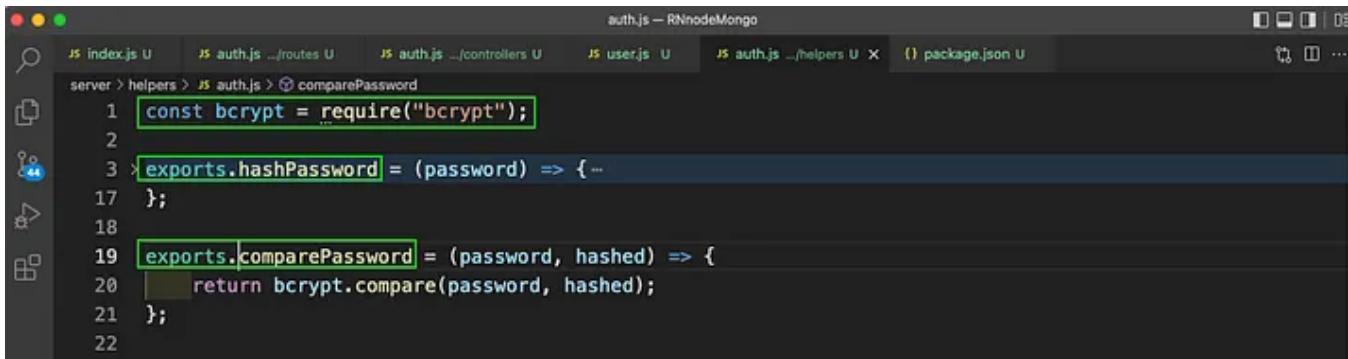
Now, we will update the imports in **user.js** for the models.



```
user.js - RNnodeMongo
JS index.js U JS auth.js .../routes U JS auth.js .../controllers U JS user.js U X JS auth.js .../helpers U {} package.json U
server > models > JS user.js > ...
1 const mongoose = require("mongoose");
2 const { Schema } = mongoose;
3
4 > const userSchema = new Schema( ...
32   },
33   { timestamps: true }
34 );
35
36 module.exports = mongoose.model("User", userSchema);
37
```

user.js

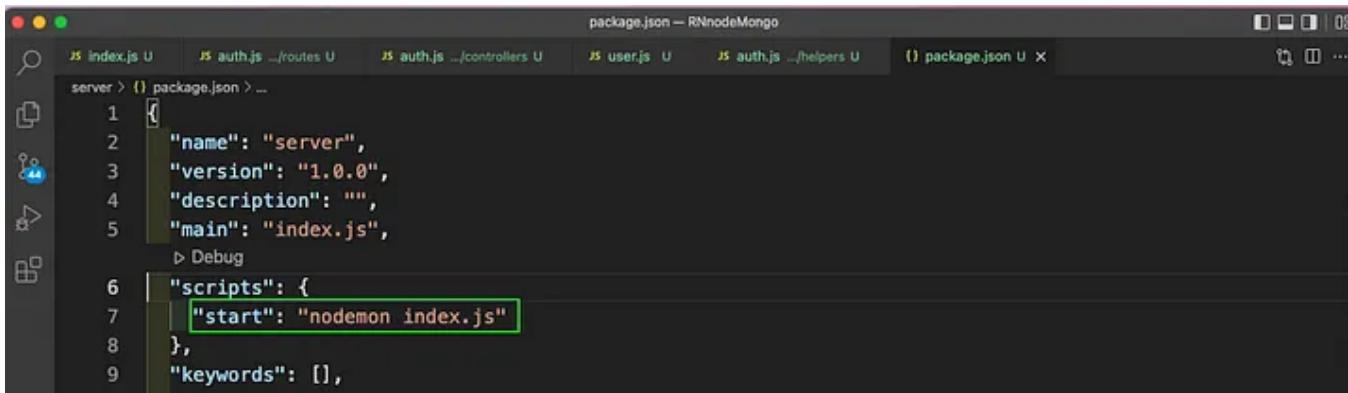
Next, we will update imports in auth.js for the helpers.



```
auth.js - RNnodeMongo
JS index.js U JS auth.js .../routes U JS auth.js .../controllers U JS user.js U JS auth.js .../helpers U {} package.json U
server > helpers > JS auth.js > comparePassword
1 const bcrypt = require("bcrypt");
2
3 > exports.hashPassword = (password) => {
4   ...
5 };
6
19 exports.comparePassword = (password, hashed) => {
20   return bcrypt.compare(password, hashed);
21 };
22
```

auth.js

We are also changing our nodemon in package.json file.



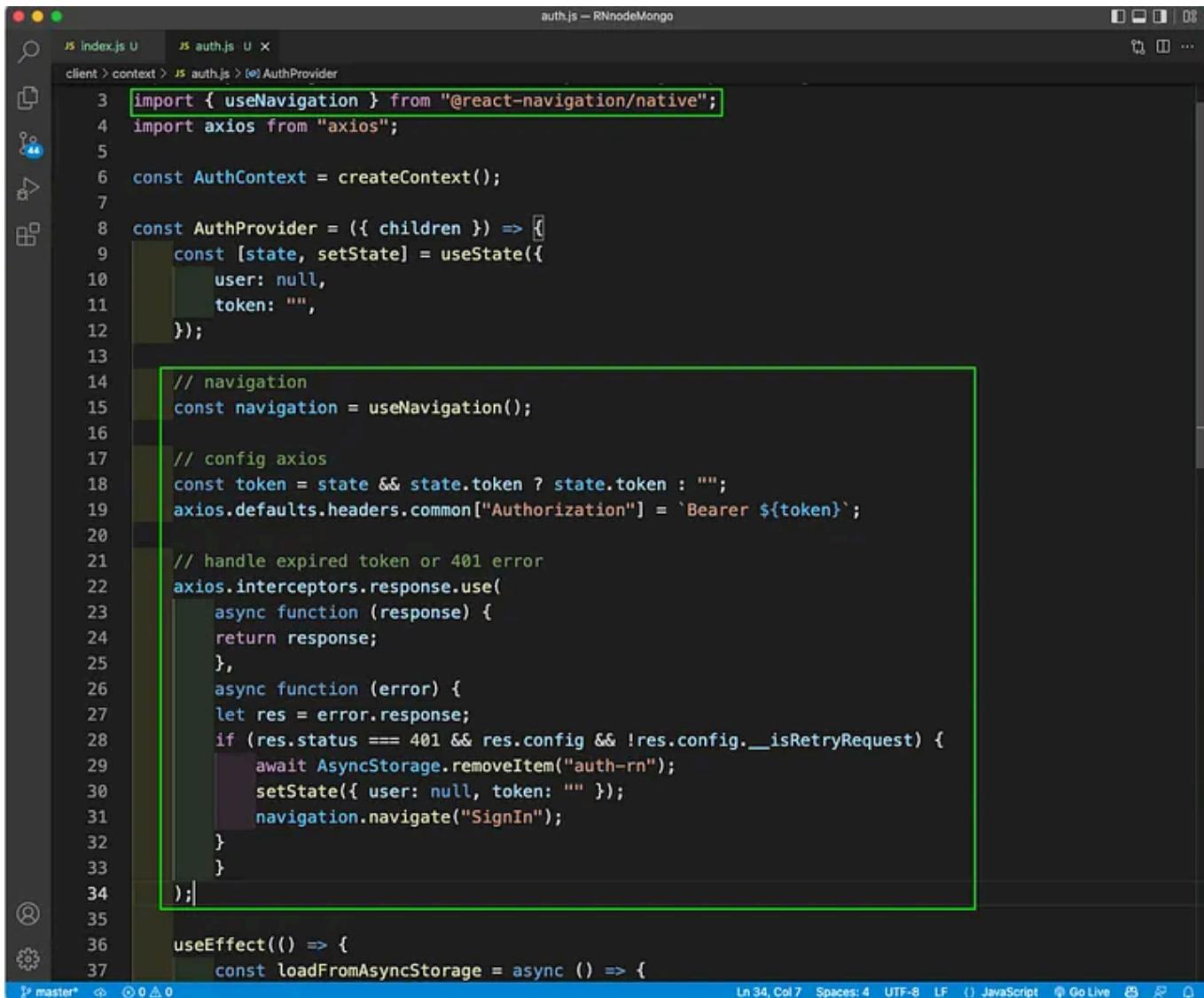
A screenshot of a code editor showing the `package.json` file for a project named "RNnodeMongo". The file contains the following JSON:

```
1  {
2    "name": "server",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "start": "nodemon index.js"
8    },
9    "keywords": []
}
```

The line `"start": "nodemon index.js"` is highlighted with a green box.

package.json

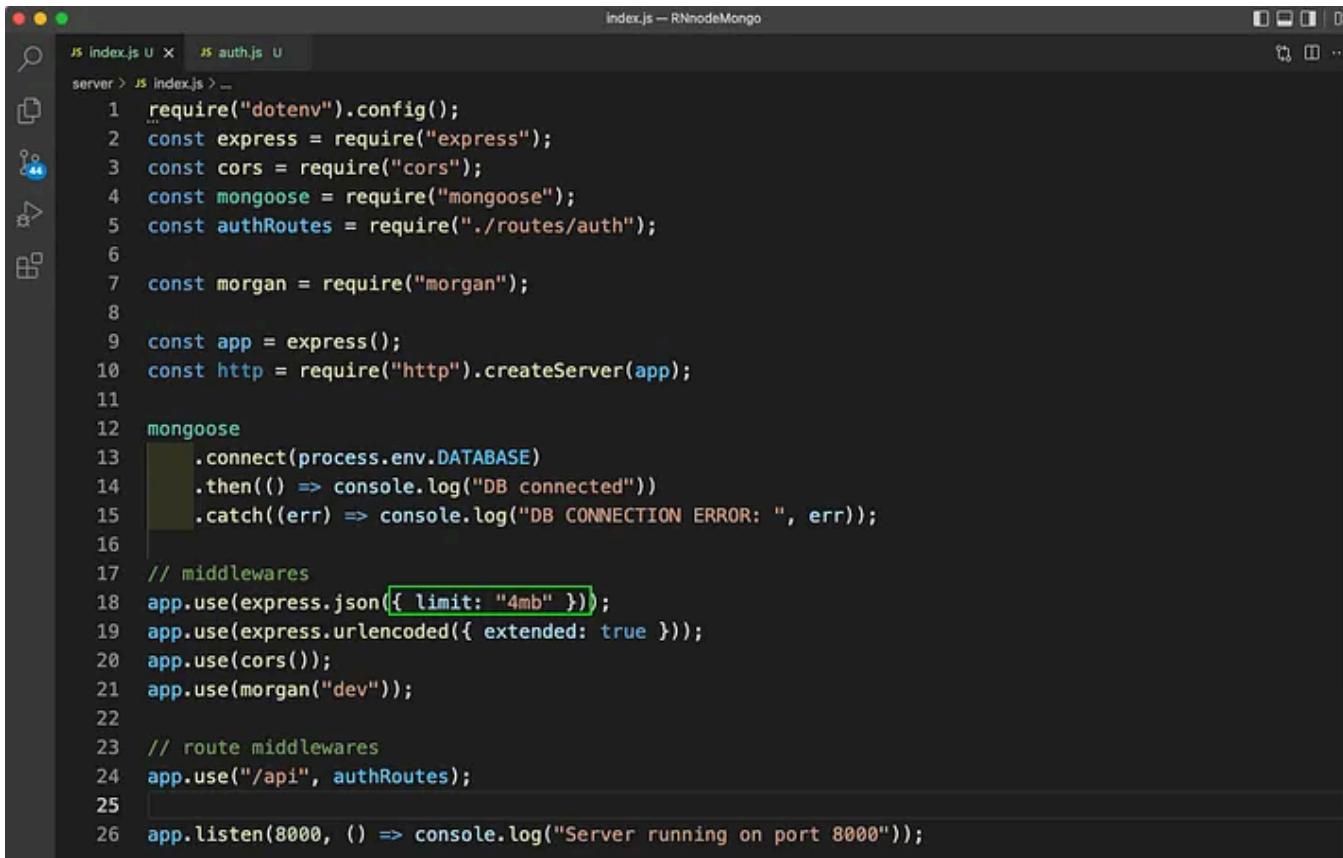
Now, in the `auth.js` in the `context` folder in client, we are adding the required header for POST request. We have also added logic to handle token expiry.



```
auth.js - RNnodeMongo
JS index.js U JS auth.js U X
client > context > JS auth.js > (e)AuthProvider
3 import { useNavigation } from "@react-navigation/native";
4 import axios from "axios";
5
6 const AuthContext = createContext();
7
8 const AuthProvider = ({ children }) => [
9   const [state, setState] = useState({
10     user: null,
11     token: "",
12   });
13
14   // navigation
15   const navigation = useNavigation();
16
17   // config axios
18   const token = state && state.token ? state.token : "";
19   axios.defaults.headers.common["Authorization"] = `Bearer ${token}`;
20
21   // handle expired token or 401 error
22   axios.interceptors.response.use(
23     async function (response) {
24       return response;
25     },
26     async function (error) {
27       let res = error.response;
28       if (res.status === 401 && res.config && !res.config.__isRetryRequest) {
29         await AsyncStorage.removeItem("auth-rn");
30         setState({ user: null, token: "" });
31         navigation.navigate("SignIn");
32       }
33     }
34   );
35
36   useEffect(() => {
37     const loadFromAsyncStorage = async () => {
Ln 34, Col 7  Spaces: 4  UTF-8  LF  ( JavaScript  Go Live
```

auth.js

We were getting an error for json request, so we have to update the limit of it in **index.js** file inside the server.

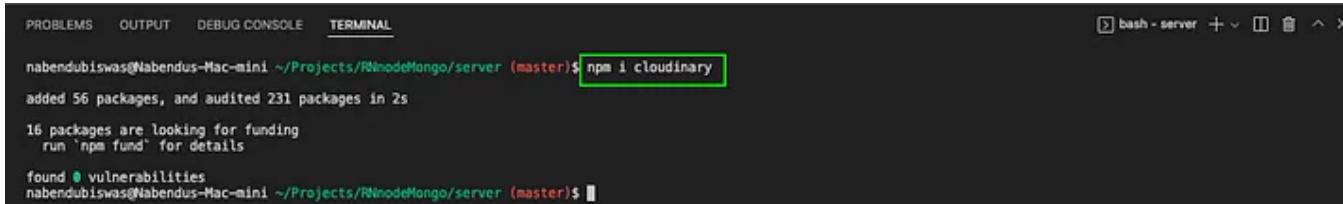


```
index.js — RNnodeMongo
JS index.js U JS auth.js U
server > JS index.js > ...
1  require("dotenv").config();
2  const express = require("express");
3  const cors = require("cors");
4  const mongoose = require("mongoose");
5  const authRoutes = require("./routes/auth");
6
7  const morgan = require("morgan");
8
9  const app = express();
10 const http = require("http").createServer(app);
11
12 mongoose
13   .connect(process.env.DATABASE)
14   .then(() => console.log("DB connected"))
15   .catch((err) => console.log("DB CONNECTION ERROR: ", err));
16
17 // middlewares
18 app.use(express.json({ limit: "4mb" }));
19 app.use(express.urlencoded({ extended: true }));
20 app.use(cors());
21 app.use(morgan("dev"));
22
23 // route middlewares
24 app.use("/api", authRoutes);
25
26 app.listen(8000, () => console.log("Server running on port 8000"));


```

index.js

Next, we will add the cloudinary package in the server.

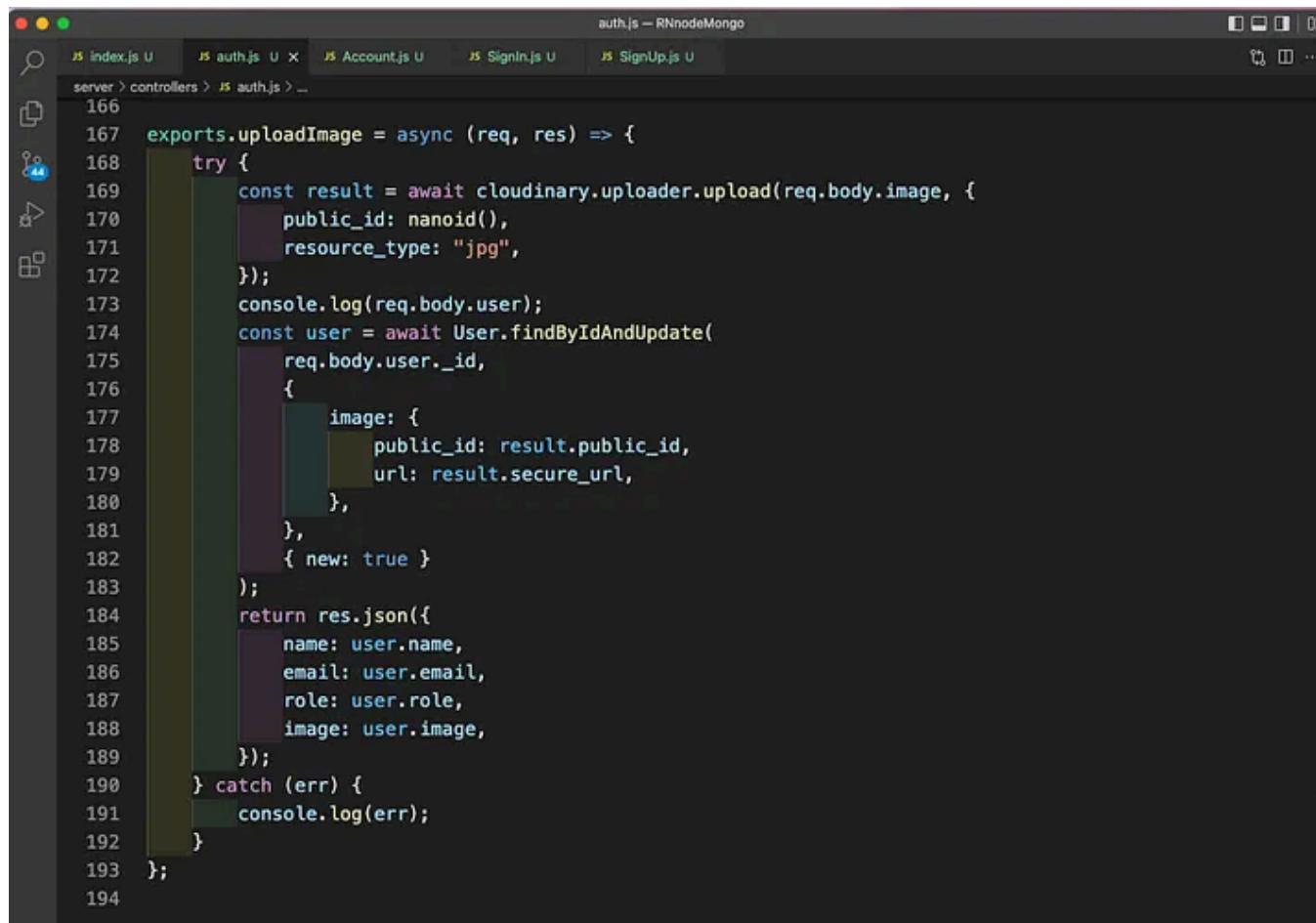


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
bash - server + × ☰ ⌂ ⌄ ^ ×
nabendubiswas@Nabendus-Mac-mini ~/Projects/RNnodeMongo/server (master)$ npm i cloudinary
added 56 packages, and audited 231 packages in 2s
16 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
nabendubiswas@Nabendus-Mac-mini ~/Projects/RNnodeMongo/server (master)$
```

Now, we will update our `uploadImage` function in `auth.js` inside controllers.

Here, we are using cloudinary to upload our image to cloudinary. Also, getting a unique id with nanoid().

After that updating the record to include the public\_id and url from cloudinary.

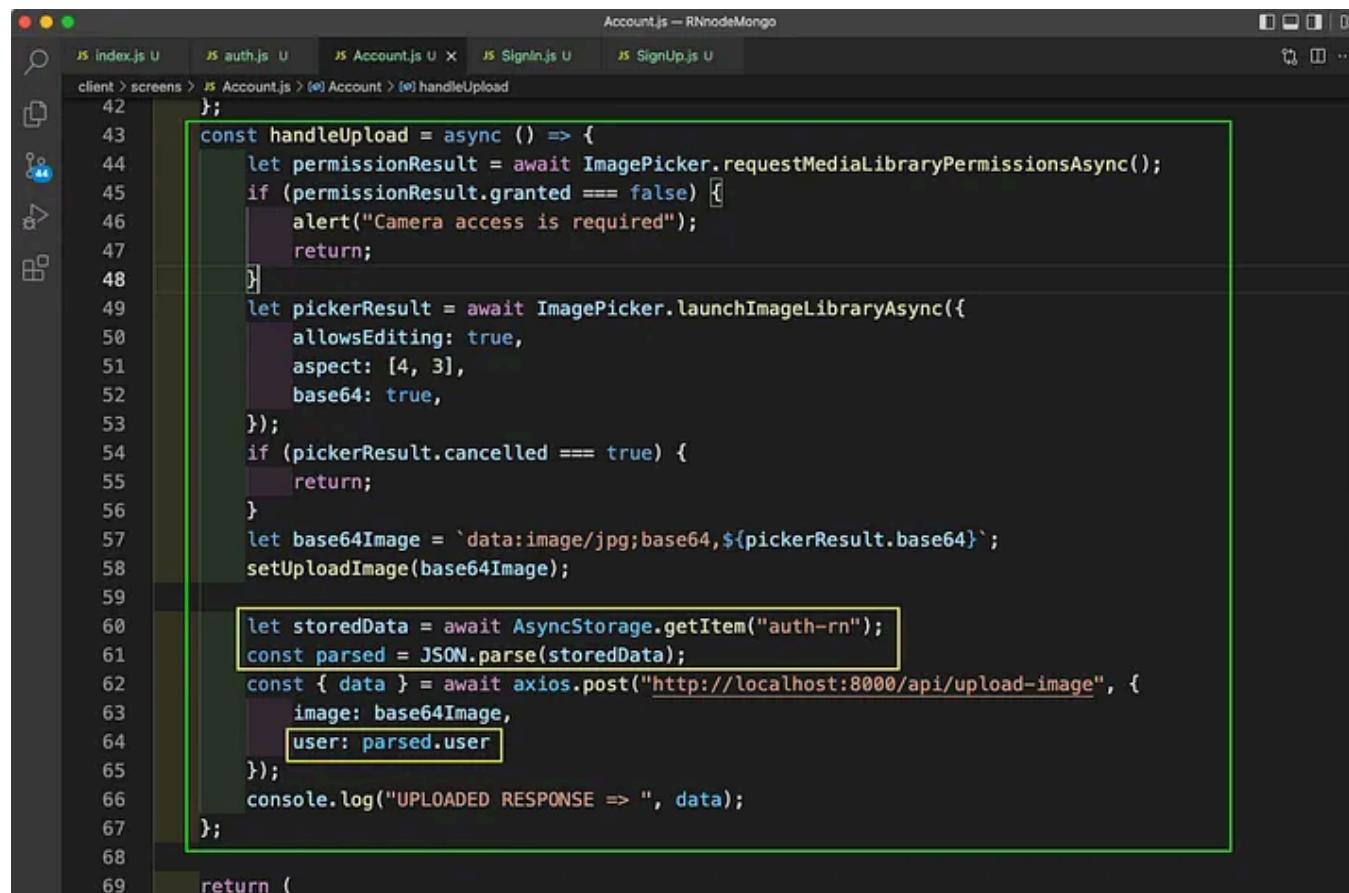


The screenshot shows a code editor window titled "auth.js – RNnodeMongo". The file contains JavaScript code for a "uploadImage" function. The code uses the cloudinary uploader to upload an image, generate a unique public\_id using nanoid(), and set the resource\_type to "jpg". It then logs the user's body and finds a user by ID to update their image field with the new public\_id and secure URL. Finally, it returns a JSON response with the user's name, email, role, and updated image object. A catch block handles any errors by logging them.

```
166
167 exports.uploadImage = async (req, res) => {
168   try {
169     const result = await cloudinary.uploader.upload(req.body.image, {
170       public_id: nanoid(),
171       resource_type: "jpg",
172     });
173     console.log(req.body.user);
174     const user = await User.findByIdAndUpdate(
175       req.body.user._id,
176       {
177         image: {
178           public_id: result.public_id,
179           url: result.secure_url,
180         },
181       },
182       { new: true }
183     );
184     return res.json({
185       name: user.name,
186       email: user.email,
187       role: user.role,
188       image: user.image,
189     });
190   } catch (err) {
191     console.log(err);
192   }
193 };
194
```

auth.js

Next, we will update our handleUpload function to get the user from the stored Async storage and pass it to upload-image endpoint in **Account.js** file.



```
JS index.js U JS auth.js U JS Account.js U X JS SignIn.js U JS SignUp.js U
client > screens > JS Account.js > (e) Account > (e) handleUpload
42   };
43   const handleUpload = async () => {
44     let permissionResult = await ImagePicker.requestMediaLibraryPermissionsAsync();
45     if (permissionResult.granted === false) {
46       alert("Camera access is required");
47       return;
48     }
49     let pickerResult = await ImagePicker.launchImageLibraryAsync({
50       allowsEditing: true,
51       aspect: [4, 3],
52       base64: true,
53     });
54     if (pickerResult.cancelled === true) {
55       return;
56     }
57     let base64Image = `data:image/jpg;base64,${pickerResult.base64}`;
58     setUploadImage(base64Image);
59
60     let storedData = await AsyncStorage.getItem("auth-rn");
61     const parsed = JSON.parse(storedData);
62     const { data } = await axios.post("http://localhost:8000/api/upload-image", {
63       image: base64Image,
64       user: parsed.user
65     });
66     console.log("UPLOADED RESPONSE => ", data);
67   };
68
69   return (
70     <View>
71       <Text>Upload Image</Text>
72       <Image source={{ uri: uploadImage }} style={ styles.image } />
73     </View>
74   );
75 }
```

Account.js

Now, when we upload an image, it is updated in the database.

The screenshot shows the MongoDB Atlas interface for a cluster named 'Cluster0'. The left sidebar has sections for Deployment, Database (selected), Data Lake, Data Services, Triggers, Data API (Preview), and Security. The main area shows 'NABENDU'S ORG - 2020-12-15 > REACT-NATIVE-MONGODBS > DATABASES'. The 'Collections' tab is selected, displaying 'myRNNodeDB.users'. The collection has 1 document. The document details are:

```
_id: ObjectId("626785551c347c711a4aceff")
name: "Nabendu"
email: "Nabendu.biswas@gmail.com"
password: "s2b$121xcRkJEKComGpG31V0LlyU.A65.YXhbtJK2MzW0SQ01tgK8jQmNba."
role: "Subscriber"
createdAt: 2022-04-26T05:38:29.265+00:00
updatedAt: 2022-04-27T08:21:46.788+00:00
v: 0
- image: Object
  public_id: "tguqttnmnekr9rjf119zk"
  url: "https://res.cloudinary.com/dxkxvfo2o/image/upload/v1651847706/tguqttnm..."
```

Next, we will update our Async storage also so that the user image is saved, even if we refresh the device.

```
const { name, email, role, image } = state.user;
setName(name);
setEmail(email);
setRole(role);
setImage(image);

}, [state]);

const handleSubmit = async () => {
};

const handleUpload = async () => {
    let permissionResult = await ImagePicker.requestMediaLibraryPermissionsAsync();
    if (permissionResult.granted === false) {
    }
    let pickerResult = await ImagePicker.launchImageLibraryAsync({
    });
    if (pickerResult.cancelled === true) {
    }
    let base64Image = `data:image/jpg;base64,${pickerResult.base64}`;
    setUploadImage(base64Image);

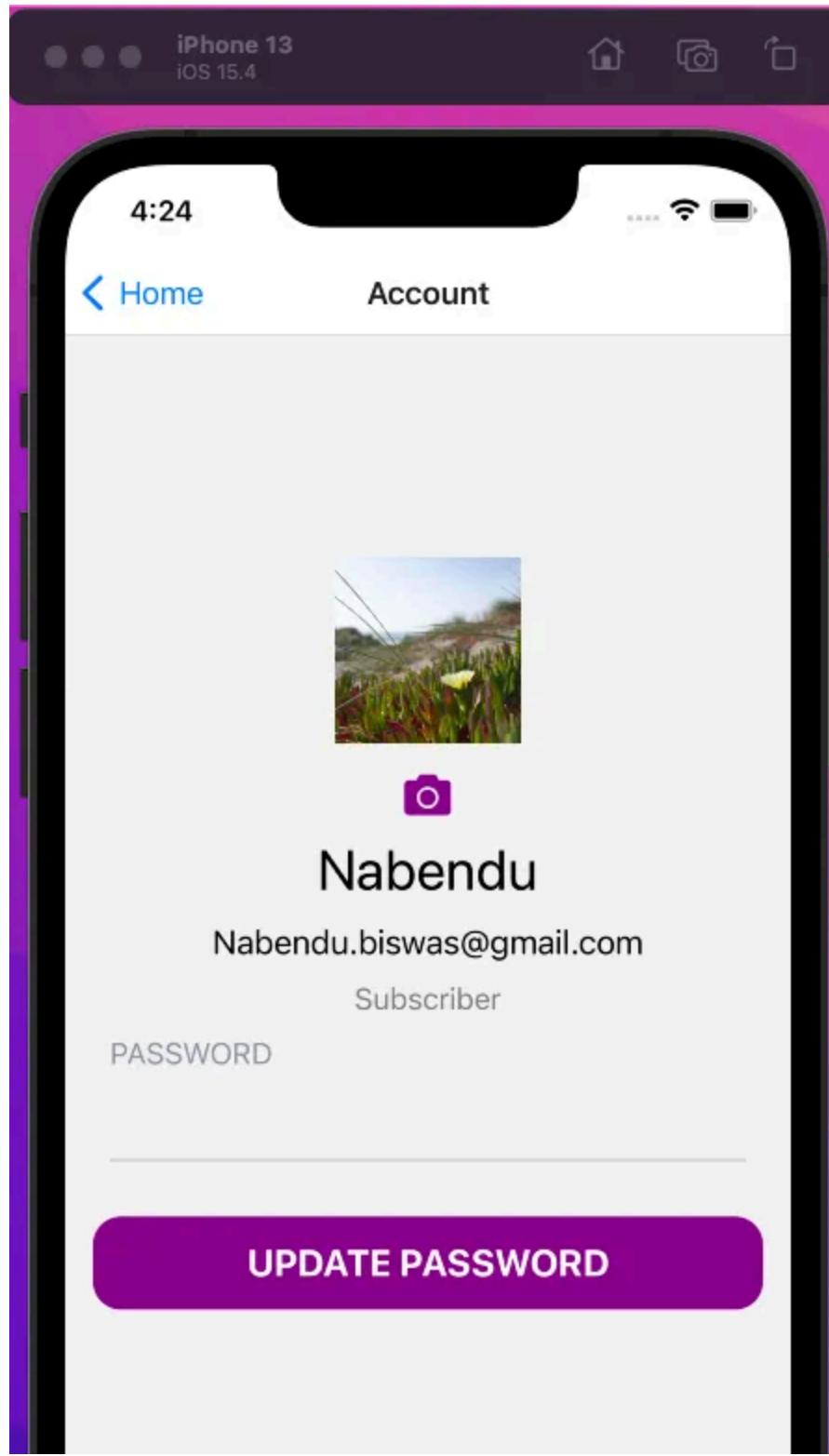
    let storedData = await AsyncStorage.getItem("auth-rn");
    const parsed = JSON.parse(storedData);
    const { data } = await axios.post("http://localhost:8000/api/upload-image", {
    });
    console.log("UPLOADED RESPONSE => ", data);
    // update async storage
    const stored = JSON.parse(await AsyncStorage.getItem("auth-rn"));
    stored.user = data;
    await AsyncStorage.setItem("auth-rn", JSON.stringify(stored));
    // update context
    setState({ ...state, user: data });
    setImage(data.image);
    alert("Profile image saved");
};

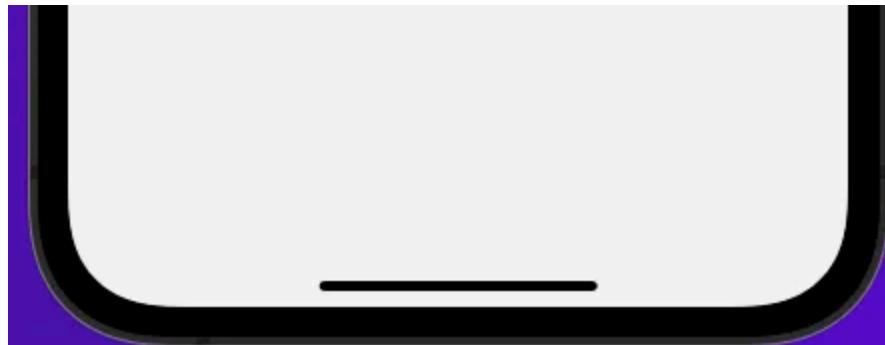
};


```

Account.js

Now, our image will be shown permanently.





This completes part-2 of the series. See you soon in part-3

React Native

React Native Development

React Native Developers

React Native App

React Native Tutorial



Written by Nabendu Biswas

Follow



903 Followers

Architect, ReactJS & Ecosystem Expert, Youtuber, Blogger

---

## More from Nabendu Biswas



 Nabendu Biswas

### React Testing with Jest and RTL in TypeScript

In this post we are going to learn to test a React app. Here, we are going to use Jest an...

15 min read · Jul 8, 2023

 17  2



 Nabendu Biswas

### NextJS 14 Blog app

In this post we will learn to create a simple blog app using the latest NextJS 14. This blo...

15 min read · Mar 2, 2024

 144  3



Nabendu Biswas

## React Native Project with NodeJS and MongoDB- Part 1

In this post we will create a major project with React Native, which will have data stored in ...

12 min read · Apr 26, 2022



92



1



[See all from Nabendu Biswas](#)



Nabendu Biswas

## TypeScript Interview Questions

Nowadays most modern ReactJS and NodeJS apps are created using TypeScript. Angular...

10 min read · Apr 14, 2024



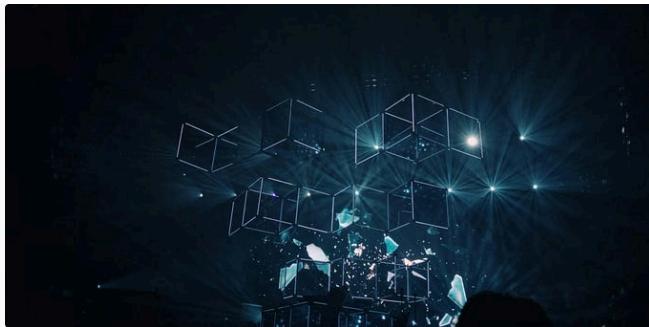
28



1



**Recommended from Medium**



 Onix React

## Release React Native 0.74.0

React Native 0.74 has been released, bringing with it a slew of updates designed to enhanc...

4 min read · Apr 23, 2024

 578



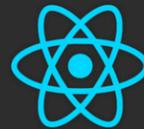
 2



1



## Building Authentication in React Native with Firebase: A Step-by-Step Guide



Mathesh Yugeswaran

 Matheshyugeswaran

## Building Authentication in React Native with Firebase: A Step-by-...

Here, we are going to delve into setting up React Native with Firebase and then setting...

4 min read · Feb 22, 2024

### Lists



#### Stories to Help You Grow as a Software Developer

19 stories · 1027 saves



 Sanjana Shivananda

## Mastering User Authentication: MERN Stack Login Page (Part 1—...)

Introduction

9 min read · Dec 19, 2023



7

 Harshit Kishor

## React Native New Architecture

React Native New Architecture vs Old Architecture

8 min read · 6 days ago



5

```
const data: {
  id: string;
  image: string;
  price: number;
}[] = await response.json();
setListProduct(data);
} catch (error) {
  console.error('Error fetching data: ', error);
}

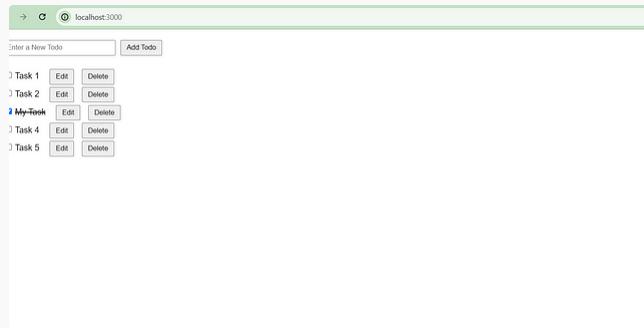
getProductList(),
[], []);

return (
<View>
```

 Charisma Kurniawan Aji

## Make your React Native code cleaner

Feeling overwhelmed by a cluttered mess of React Native code? As your app expands, so...



 Maitrik Thakkar

## Simple Todo App using Redux Toolkit

Install React Project

3 min read · Mar 8, 2024

3 min read · Feb 25, 2024



46



1



2



See more recommendations

[Help](#) [Status](#) [About](#) [Careers](#) [Blog](#) [Privacy](#) [Terms](#) [Text to speech](#) [Teams](#)