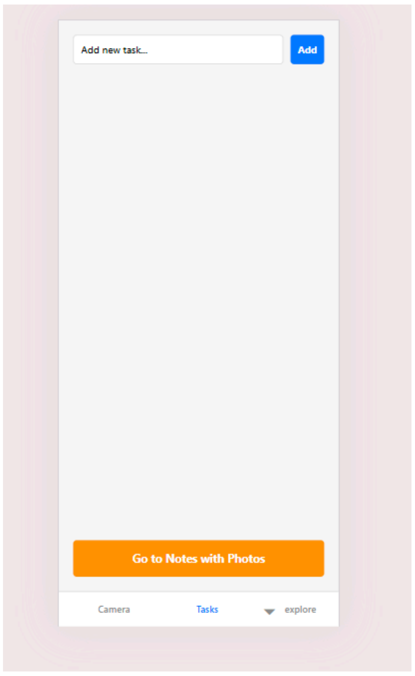
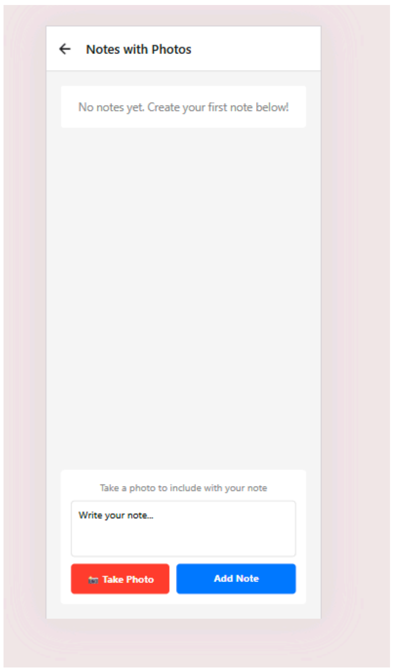
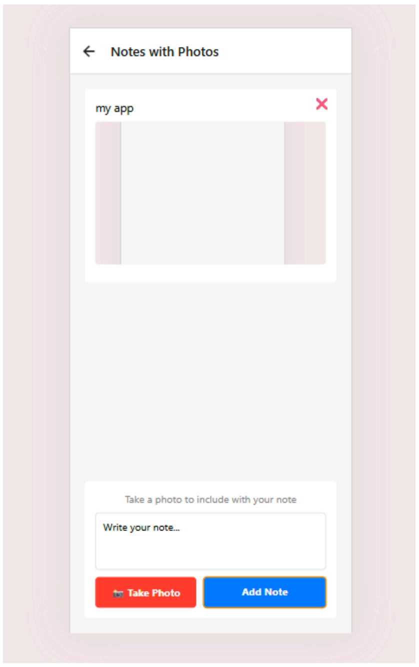
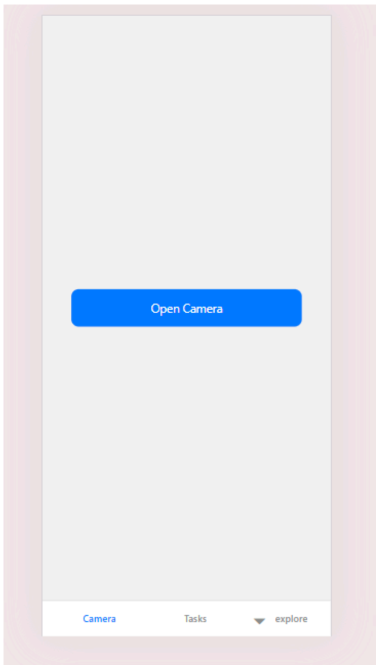




No notes yet. Create your first note below!





Current State of the Application

This project started as two separate apps – one for taking photos and one for creating simple tasks. I have now combined them into a single app called **Forgetful**.

The basic functionality is working:

- The user can create a task and take a photo.
- The app combines the camera and task features into one flow.

Right now, the app is simple and not yet fully designed. It works, but it's not perfect. There are bugs and parts that still need improvement.

Next Steps

The main goal now is to:

- Connect and configure local database storage (e.g., AsyncStorage or expo-file-system),
- Make sure tasks and photos are saved and restored properly between sessions,
- Improve the overall look and feel of the app (layout, styling, navigation),
- Fix issues with data disappearing when switching screens.

Current Issue

Photos Disappear When Navigating Between Screens

Currently, when a user takes a photo and navigates away from the screen, the image disappears. This means the task and its associated photo are not stored or retrieved properly, and are lost when switching views or restarting the app.

The application does not yet implement persistent storage. Photos and task data are only held temporarily in state, which is cleared when the component unmounts or when the app reloads.

Planned Solution:
Integrate local storage

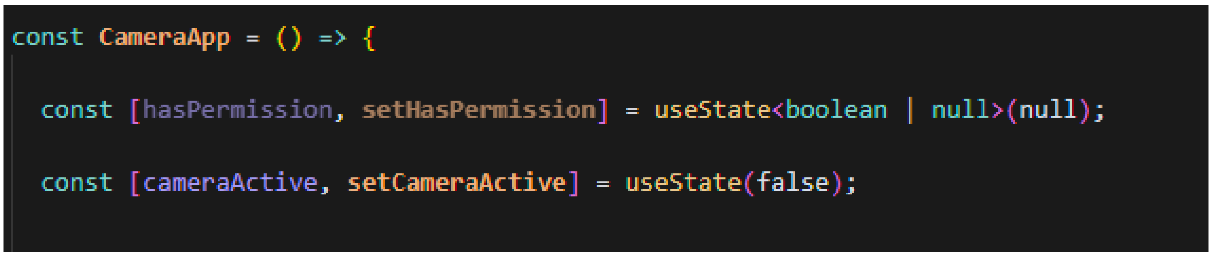
Issue I had With opening the camera

While combining the camera functionality with the task part of the app, I ran into a problem where clicking the button to open the camera didn't seem to do anything. The screen didn't crash, but the camera view didn't show up. I expected it to change state and render the preview, but nothing happened.

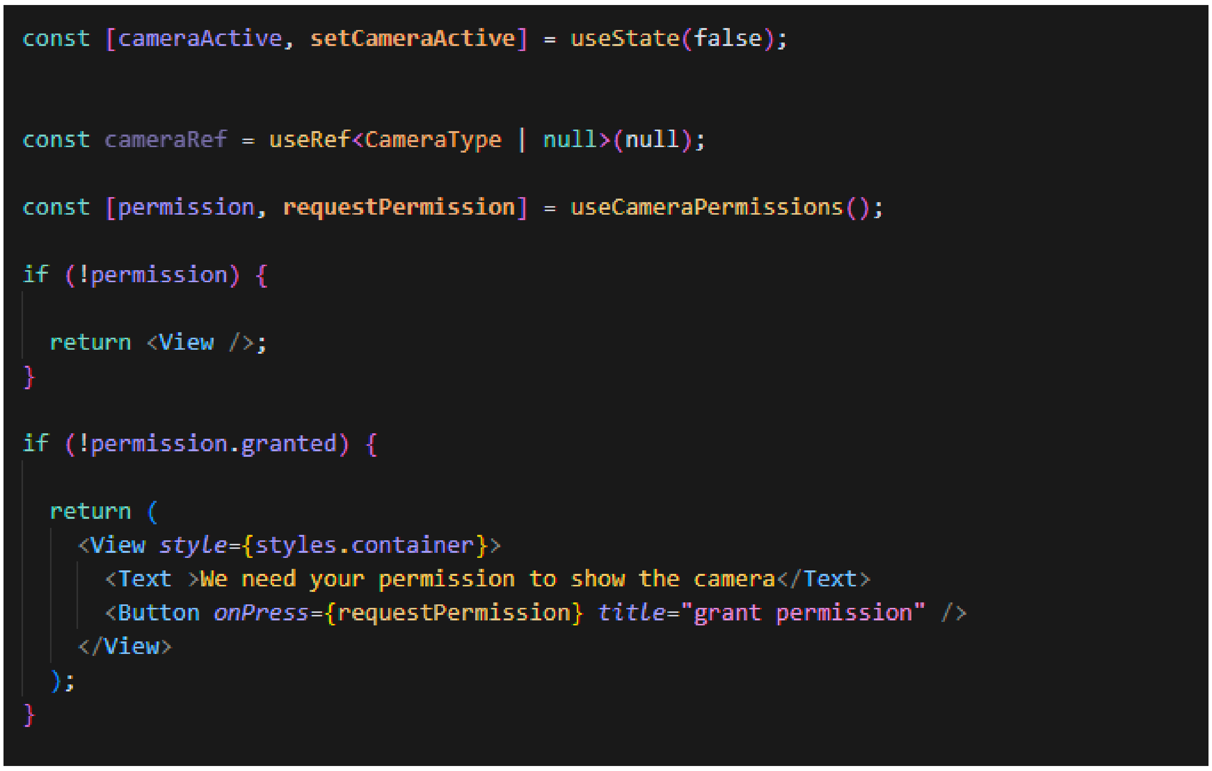
At first I thought it might be a problem with how I was handling state, or maybe some mistake in how I used `useRef`. But after checking things step by step, I realized the issue was with the camera permissions. The app was trying to render the `CameraView` before the user had actually granted access to the camera, so it simply didn't show anything.

I fixed it by making sure the camera is only shown **after** permission is granted, and only if the user clicks the button to activate it. Below are the fragments of code that currently handle this correctly.

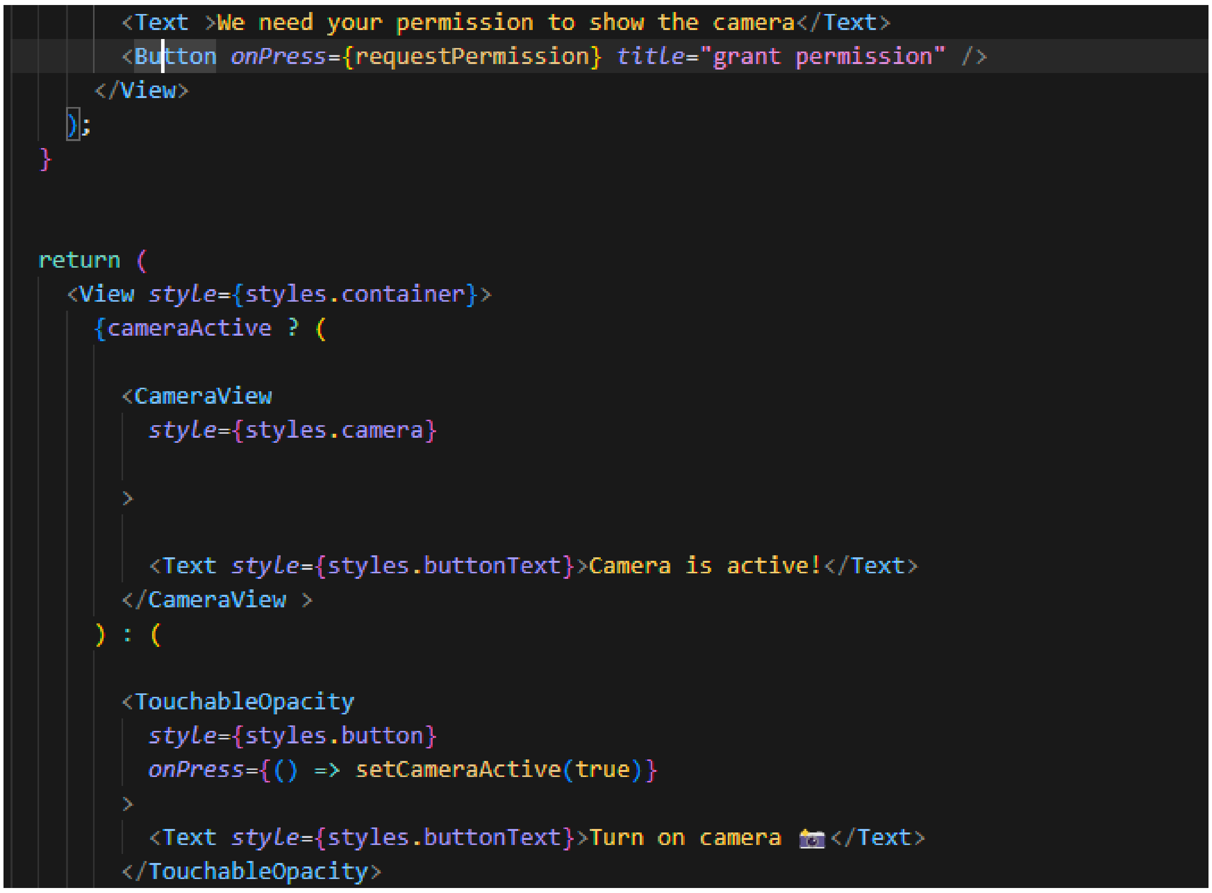
Once I put this together, everything started working properly. The camera view shows only after the user has granted permission and presses the button. Before that, the app just shows a simple message with the option to grant access.



```
const CameraApp = () => {

  const [hasPermission, setHasPermission] = useState<boolean | null>(null);

  const [cameraActive, setCameraActive] = useState(false);
```



```
const [cameraActive, setCameraActive] = useState(false);


const cameraRef = useRef<CameraType | null>(null);

const [permission, requestPermission] = useCameraPermissions();

if (!permission) {

  return <View />;
}

if (!permission.granted) {

  return (
    <View style={styles.container}>
      <Text >We need your permission to show the camera</Text>
      <Button onPress={requestPermission} title="grant permission" />
    </View>
  );
}
```



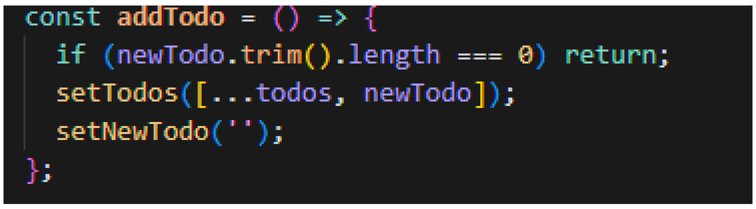
```
      <Text >We need your permission to show the camera</Text>
      <Button onPress={requestPermission} title="grant permission" />
    </View>
  );
}


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

      <CameraView
        style={styles.camera}


      >

        <Text style={styles.buttonText}>Camera is active!</Text>
      </CameraView >
    ) : (

      <TouchableOpacity
        style={styles.button}
        onPress={() => setCameraActive(true)}
      >
        <Text style={styles.buttonText}>Turn on camera 📷</Text>
      </TouchableOpacity>
```

Another Issue I had was adding empty tasks

While building the task screen, I ran into a problem where I kept accidentally adding empty tasks. Sometimes I pressed the "Add" button too quickly without typing anything, and a blank task would still appear in the list. It didn't break anything, but it looked messy and made the app feel buggy.

At first I wasn't sure why this was happening, because I assumed the input would just reject empty strings. But then I checked the code and realized I hadn't added any condition to prevent that.

I fixed it by adding a simple check to ignore blank inputs. Now, if I press the button with an empty string or just spaces, nothing gets added. Here's the code that currently handles that:



```
const addTodo = () => {
  if (newTodo.trim().length === 0) return;
  setTodos([...todos, newTodo]);
  setNewTodo('');
};
```

This little change made the app feel much better right away. Now I don't have to worry about filling the list with empty entries by accident.

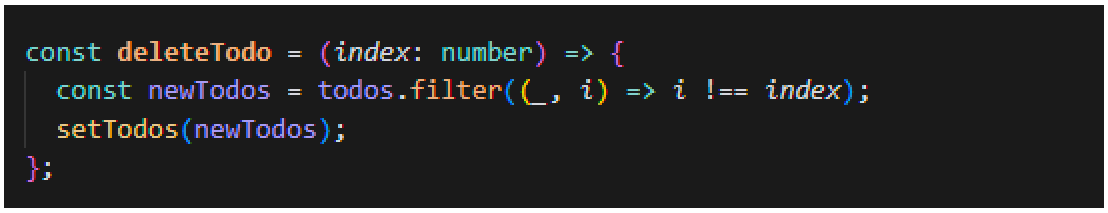
Problem with Deleting the Wrong Task

While testing the to-do list screen, I noticed that sometimes when I clicked the ❌ button to delete a task, it felt like the wrong one was removed. It didn't always happen, but it made me unsure if the delete logic was working correctly.

At first I thought maybe the list wasn't updating fast enough, or React Native was reusing some of the items. But then I realized the real issue was probably in how I was handling the task list and passing the index.

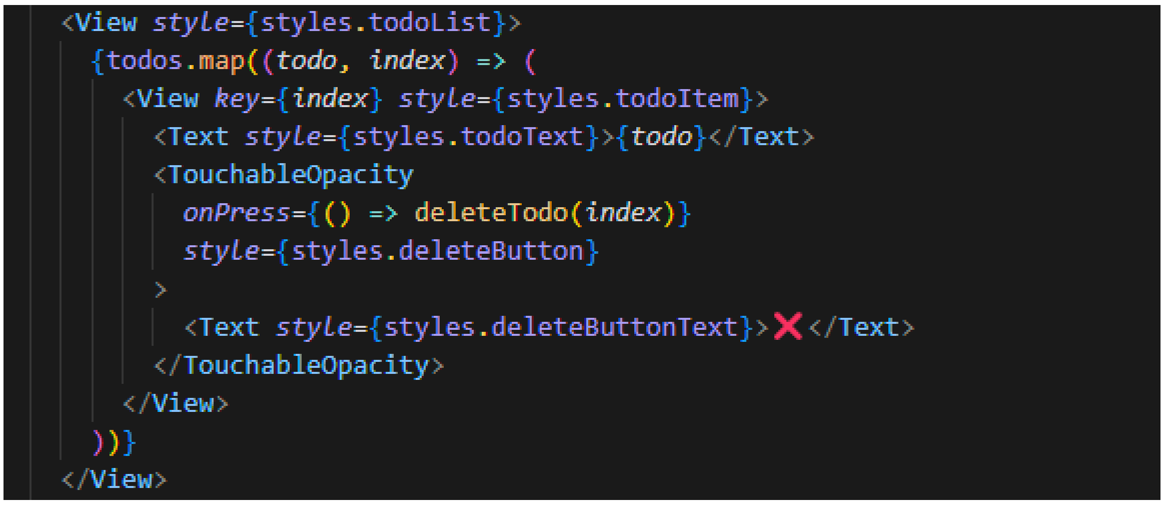
In the delete function, I filter out the selected task by its index. This works as long as the indexes are stable and everything renders in order. After some testing, I saw that it actually works fine – I just wasn't updating the UI in a noticeable way, so it **looked like** the wrong item disappeared when in fact it didn't.

Here's the part of the code that does the job:



```
const deleteTodo = (index: number) => {
  const newTodos = todos.filter((_, i) => i !== index);
  setTodos(newTodos);
};
```

And in the UI:



```
  <View style={styles.todoList}>
    {todos.map((todo, index) => (
      <View key={index} style={styles.todoItem}>
        <Text style={styles.todoText}>{todo}</Text>
        <TouchableOpacity
          onPress={() => deleteTodo(index)}
          style={styles.deleteButton}
        >
          <Text style={styles.deleteButtonText}>❌</Text>
        </TouchableOpacity>
      </View>
    ))}
  </View>
```

So in the end, the logic was fine it just needed better visual feedback. Later I might add a quick animation or a toast notification to confirm the deletion happened.