Lab: Enahnce SQLite Database in a React Native App

Estimated time needed: 30 minutes

Welcome to this instructional lab, where you will learn to set up and integrate an SQLite database into a React Native application. You will create a simple **Reminder Notes** app that allows users to store, retrieve, and delete notes.

Prerequisites

- You must be familiar with JavaScript and some basic SQL commands.
- You must have an expo account created to test this lab on an actual device. You can create it by going to this link.
- You must have **Expo** app installed on your phone to try this app on a real device.

Objectives

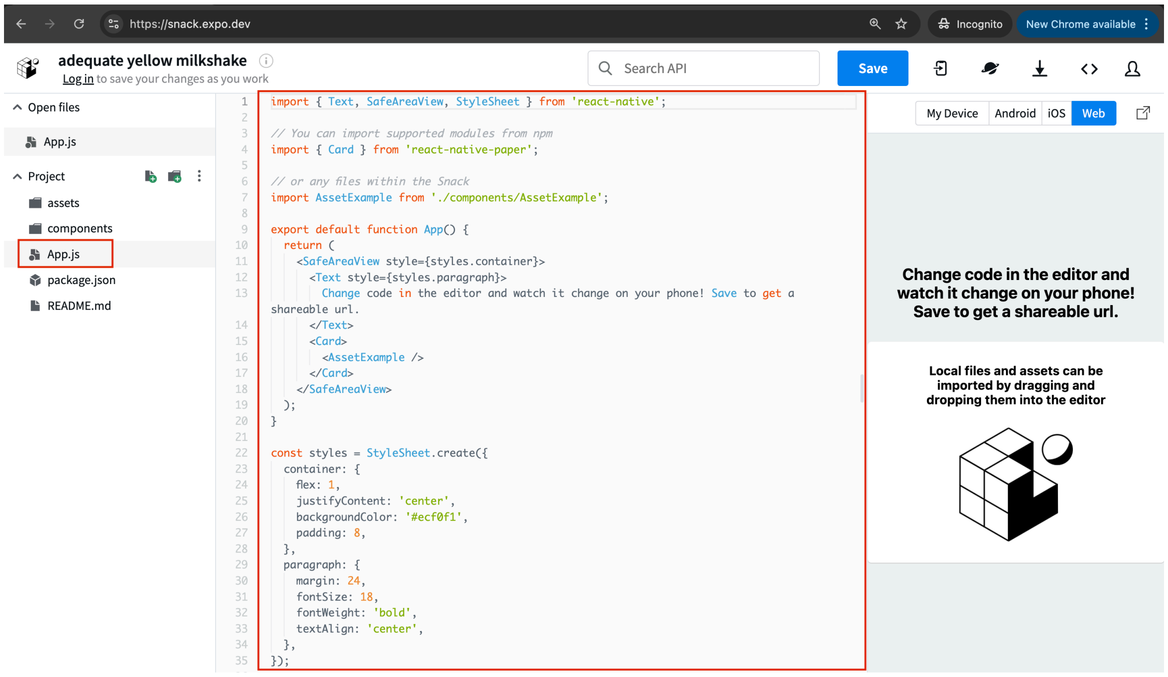
After completing this lab, you will be able to:

- Apply in-memory storage in a React Native app using SQLite
- Create a React Native application that will allow users to store, retrieve and delete notes
- Manage the database operations from within the React Native app in real-time

Step 1: Create and set up the app

1. Open this link to use `snack expo` to create a React Native app. You will be using this environment as the web browser doesn't support React Native SQLite unless explicitly created as a web app. You will not need to sign in. However, if you would like to save your app, it is recommended that you sign in.

2. You will see that you are provided with some default application code when you open. The file that you see is `App.js`. This is what you will be using to create the app.



adequate yellow milkshake

Log in to save your changes as you work

```
import { Text, SafeAreaView, StyleSheet } from 'react-native';

// You can import supported modules from npm
import { Card } from 'react-native-paper';

// or any files within the Snack
import AssetExample from './components/AssetExample';

export default function App() {
  return (
    <SafeAreaView style={styles.container}>
      <Text style={styles.paragraph}>
        Change code in the editor and watch it change on your phone! Save to get a shareable url.
      </Text>
      <Card>
        <AssetExample />
      </Card>
    </SafeAreaView>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    backgroundColor: '#ecf0f1',
    padding: 8,
  },
  paragraph: {
    margin: 24,
    fontSize: 18,
    fontWeight: 'bold',
    textAlign: 'center',
  },
});
```

Change code in the editor and watch it change on your phone! Save to get a shareable url.

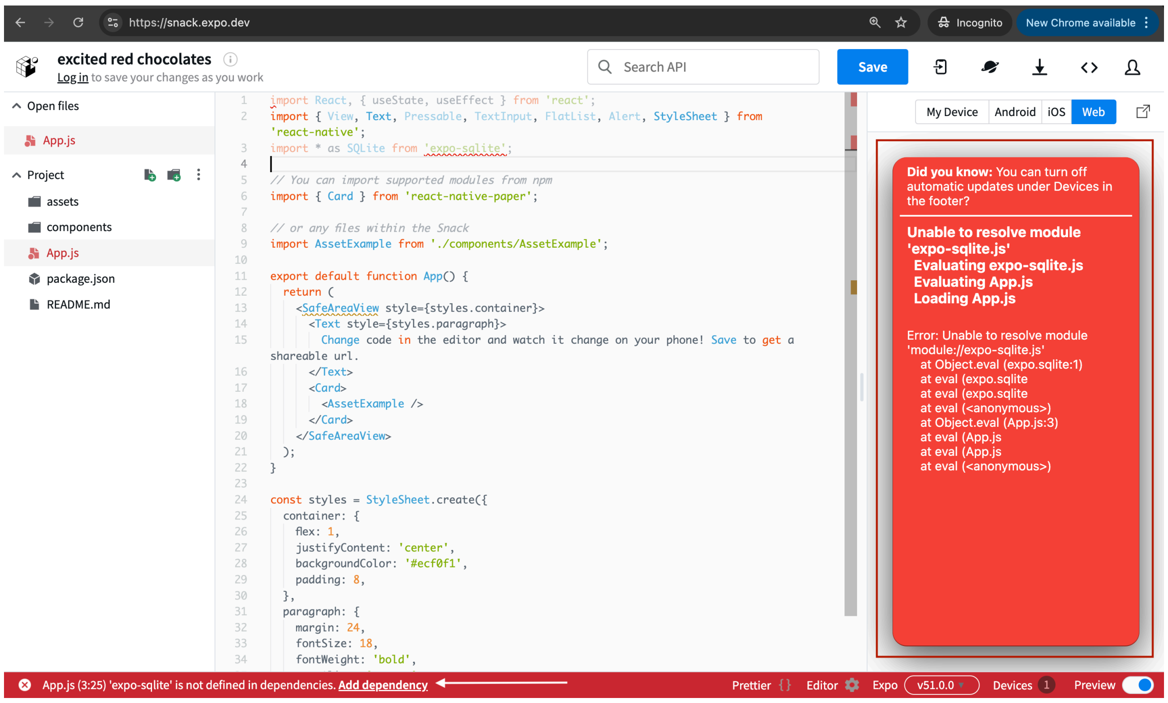
Local files and assets can be imported by dragging and dropping them into the editor

Step 2: Edit the application

1. Replace the import section of the `App.js` with the following content.

```
import React, { useState, useEffect } from 'react';
import { View, Text, Pressable, TextInput, FlatList, Alert, StyleSheet } from 'react-native';
import * as SQLite from 'expo-sqlite';
```

2. The moment you add the imports, you will get an error as the `expo-sqlite` dependency is not installed by default. Click `Add Dependency` on the bottom of the screen to add the package dependency.



excited red chocolates

Log in to save your changes as you work

```
import React, { useState, useEffect } from 'react';
import { View, Text, Pressable, TextInput, FlatList, Alert, StyleSheet } from
'react-native';
import * as SQLite from 'expo-sqlite';

// You can import supported modules from npm
import { Card } from 'react-native-paper';

// or any files within the Snack
import AssetExample from './components/AssetExample';

export default function App() {
  return (
    <SafeAreaView style={styles.container}>
      <Text style={styles.paragraph}>
        Change code in the editor and watch it change on your phone! Save to get a shareable url.
      </Text>
      <Card>
        <AssetExample />
      </Card>
    </SafeAreaView>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    backgroundColor: '#ecf0f1',
    padding: 8,
  },
  paragraph: {
    margin: 24,
    fontSize: 18,
    fontWeight: 'bold',
```

Did you know: You can turn off automatic updates under Devices in the footer?

```
Unable to resolve module 'expo-sqlite.js'
  Evaluating expo-sqlite.js
  Evaluating App.js
  Loading App.js

Error: Unable to resolve module 'module://expo-sqlite.js'
  at Object.eval (expo.sqlite:1)
  at eval (expo.sqlite
  at eval (expo.sqlite
  at eval (<anonymous>)
  at Object.eval (App.js:3)
  at eval (App.js
  at eval (App.js
  at eval (<anonymous>)
```

App.js (3:25) 'expo-sqlite' is not defined in dependencies. Add dependency

3. Remove the content of the `App()` method as you will recreate the whole method for the `Reminder` app. It should appear as below.

```
export default function App() {
}
```

4. Add the following hooks to handle the db, the note that is being added and the list of notes inside the `App()` function.

```
const [note, setNote] = useState('');
const [notes, setNotes] = useState([]);
const [db,setDb] = useState(null);
```

5. Now, you will add some asynchronous functions to load the database, create the table, insert data into the table, delete data from the table and fetch the data that exists in the table with the `App()` function. You will use `useEffect` to populate the initial values. First, add the loadDB function given below. This function will load open the database and create the `Notes` table if one doesn't exist, which is the case when you first open the app.

```
const loadDB = async ()=>{
  setDb(await SQLite.openDatabaseAsync('notes.db'))
  if(db != null) {
    await db.runAsync(
      'CREATE TABLE IF NOT EXISTS Notes (id INTEGER PRIMARY KEY AUTOINCREMENT, content TEXT);',
      [],
      () => console.log('Table created successfully'),
      (_, error) => console.log('Error in creating table:', error)
    );
}
}
```

6. You will next add the `useEffect` which will load the initial list if it exists.

```
useEffect(() => {
  loadDB().then(()=>{
    fetchNotes();
  });
}, [note, notes]);
```

7. Then you will add the `fetchNotes` function which will repopulate the list of notes on the screen.

```
const fetchNotes = async () => {
    if(db != null) {
      const allRows = await db.getAllAsync('SELECT * FROM Notes');
      let updateNotes = []
      for (const row of allRows) {
        updateNotes.push({"id":row.id, "content":row.content});
      }
      setNotes(updateNotes);
    }
};
```

8. Then, you will add the `addNote` function which allows the app user to add the notes.

```
const addNote = async () => {
  if (!note.trim()) {
    Alert.alert('Please enter a note');
    return;
  }
  db.runAsync('INSERT INTO Notes (content) VALUES (?);',
      [note]).then((output)=>{
        fetchNotes();
      });
  setNote('');
};
```

9. You will now add `deleteNote` that allows the app user to delete the notes that are not required anymore.

```
const deleteNote = (id) => {
      db.runAsync(
        'DELETE FROM Notes WHERE id = ?;',
        id,
        (_, result) => {
          console.log('Note deleted:', result);
        },
        (_, error) => console.log('Error deleting note:', error)
      ).then(async ()=>{
        await fetchNotes();
      })
  };
```

10. Now, you need to add the UI to the view to be rendered on the app and the respective style. Include the following code, after the asynchrounous functions you added earlier.

```
return (
    <View style={styles.container}>
      <Text style={styles.title}>Reminder Notes</Text>
      <TextInput
        style={styles.input}
        placeholder="Write a note"
        value={note}
        onChangeText={setNote}
      />
      <Pressable  style={styles.button} onPressIn={addNote}>
      <Text style={styles.buttonText}>Add Note</Text>
      </Pressable>
      <FlatList
        data={notes}
        keyExtractor={(item) => item.id.toString()}
        renderItem={({ item }) => (
          <View style={styles.noteContainer}>
            <Text style={styles.noteText}>{item.content}</Text>
            <Pressable style={styles.button} onPressIn={() => deleteNote(item.id)}>
              <Text style={styles.buttonText}>Delete</Text>
            </Pressable>
          </View>
        )}
      />
    </View>
  );
```

11. Finally, after the `App()` function, you will define the styles which were applied to the components followed by `export default App;` to use the app component in other files when needed.

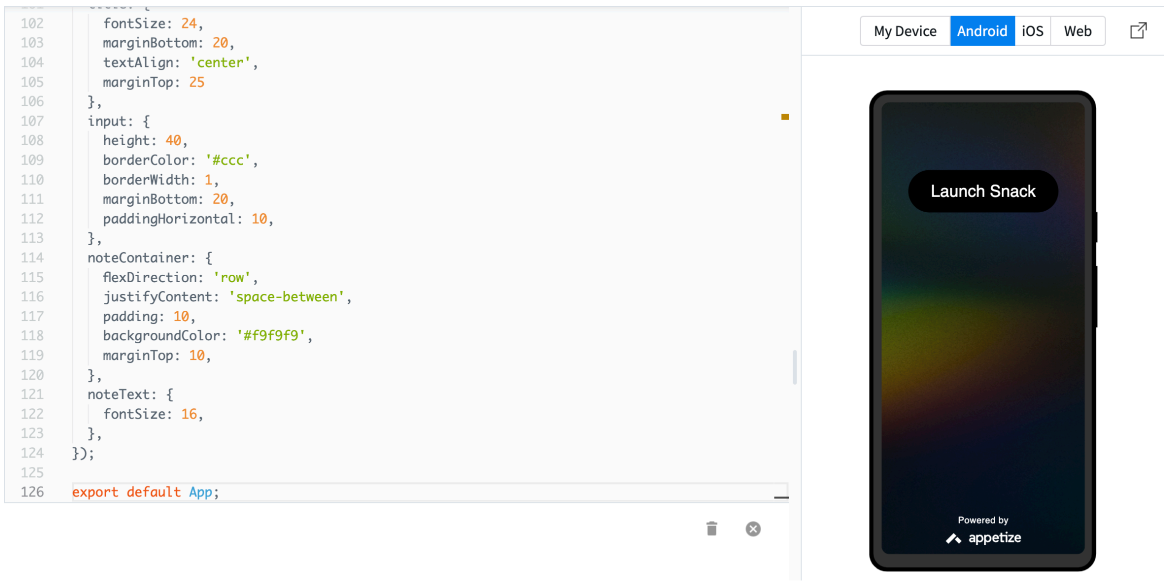
```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 20,
    justifyContent: 'center',
    backgroundColor: '#fff',
  },
  title: {
    fontSize: 24,
    marginBottom: 20,
    textAlign: 'center',
    marginTop: 25
  },
  input: {
    height: 40,
    borderColor: '#ccc',
    borderWidth: 1,
    marginBottom: 20,
    paddingHorizontal: 10,
  },
  noteContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    padding: 10,
    backgroundColor: '#f9f9f9',
    marginTop: 10,
  },
  noteText: {
    fontSize: 16,
  },
  button: {
    backgroundColor: 'purple',
    color:'white',
    alignItems:'center',
    justifyContent: 'center',
    height: 50,
    margin:10
  },
  buttonText: {
    color:'white',
    fontSize:20,
    alignSelf: 'center'
  }
});
export default App;
```

12. When completed, your `App.js` should have the following content.

```
import React, { useState, useEffect } from 'react';
import { View, Text, Pressable, TextInput, FlatList, Alert, StyleSheet } from 'react-native';
import * as SQLite from 'expo-sqlite';
const App = () => {
  const [note, setNote] = useState('');
  const [notes, setNotes] = useState([]);
  const [db,setDb] = useState(null);
  const loadDB = async ()=>{
    setDb(await SQLite.openDatabaseAsync('notes.db'))
    if(db != null) {
      await db.runAsync(
        'CREATE TABLE IF NOT EXISTS Notes (id INTEGER PRIMARY KEY AUTOINCREMENT, content TEXT);',
        [],
        () => console.log('Table created successfully'),
        (_, error) => console.log('Error in creating table:', error)
      );
  }
  }
  useEffect(() => {
    loadDB().then(()=>{
      fetchNotes();
    });
  }, [note, notes]);
  const fetchNotes = async () => {
      if(db != null) {
        const allRows = await db.getAllAsync('SELECT * FROM Notes');
        let updateNotes = []
        for (const row of allRows) {
          updateNotes.push({"id":row.id, "content":row.content});
        }
        setNotes(updateNotes);
      }
  };
  const addNote = async () => {
    if (!note.trim()) {
      Alert.alert('Please enter a note');
      return;
    }
    db.runAsync('INSERT INTO Notes (content) VALUES (?);',
        [note]).then((output)=>{
          fetchNotes();
        });
    setNote('');
  };
  const deleteNote = (id) => {
      db.runAsync(
        'DELETE FROM Notes WHERE id = ?;',
        id,
        (_, result) => {
          console.log('Note deleted:', result);
        },
        (_, error) => console.log('Error deleting note:', error)
      ).then(async ()=>{
        await fetchNotes();
      })
  };
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Reminder Notes</Text>
      <TextInput
        style={styles.input}
        placeholder="Write a note"
        value={note}
        onChangeText={setNote}
      />
      <Pressable  style={styles.button} onPressIn={addNote}>
      <Text style={styles.buttonText}>Add Note</Text>
      </Pressable>
      <FlatList
        data={notes}
        keyExtractor={(item) => item.id.toString()}
        renderItem={({ item }) => (
          <View style={styles.noteContainer}>
            <Text style={styles.noteText}>{item.content}</Text>
            <Pressable style={styles.button} onPressIn={() => deleteNote(item.id)}>
              <Text style={styles.buttonText}>Delete</Text>
            </Pressable>
          </View>
        )}
      />
    </View>
  );
};
const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 20,
    justifyContent: 'center',
    backgroundColor: '#fff',
  },
  title: {
    fontSize: 24,
    marginBottom: 20,
    textAlign: 'center',
    marginTop: 25
  },
  input: {
    height: 40,
    borderColor: '#ccc',
    borderWidth: 1,
    marginBottom: 20,
    paddingHorizontal: 10,
  },
  noteContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    padding: 10,
    backgroundColor: '#f9f9f9',
    marginTop: 10,
  },
  noteText: {
    fontSize: 16,
  },
  button: {
    backgroundColor: 'purple',
    color:'white',
    alignItems:'center',
    justifyContent: 'center',
    height: 50,
    margin:10
  },
  buttonText: {
    color:'white',
    fontSize:20,
    alignSelf: 'center'
  }
});
export default App;
```

Step 3: Testing the app

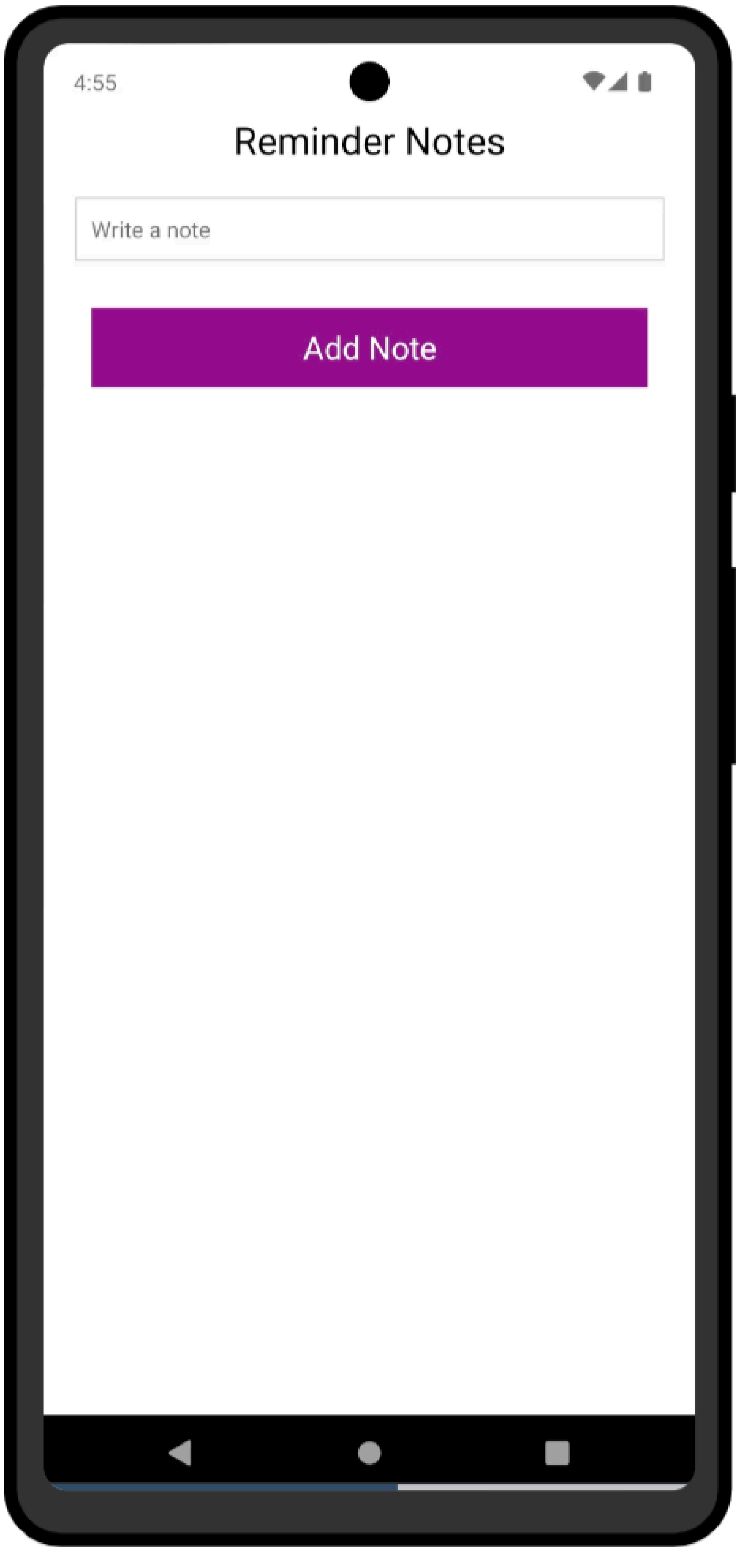
1. You can now test the app in the simulator, by choosing `Android` from display options on the right side and click `Launch Snack`.



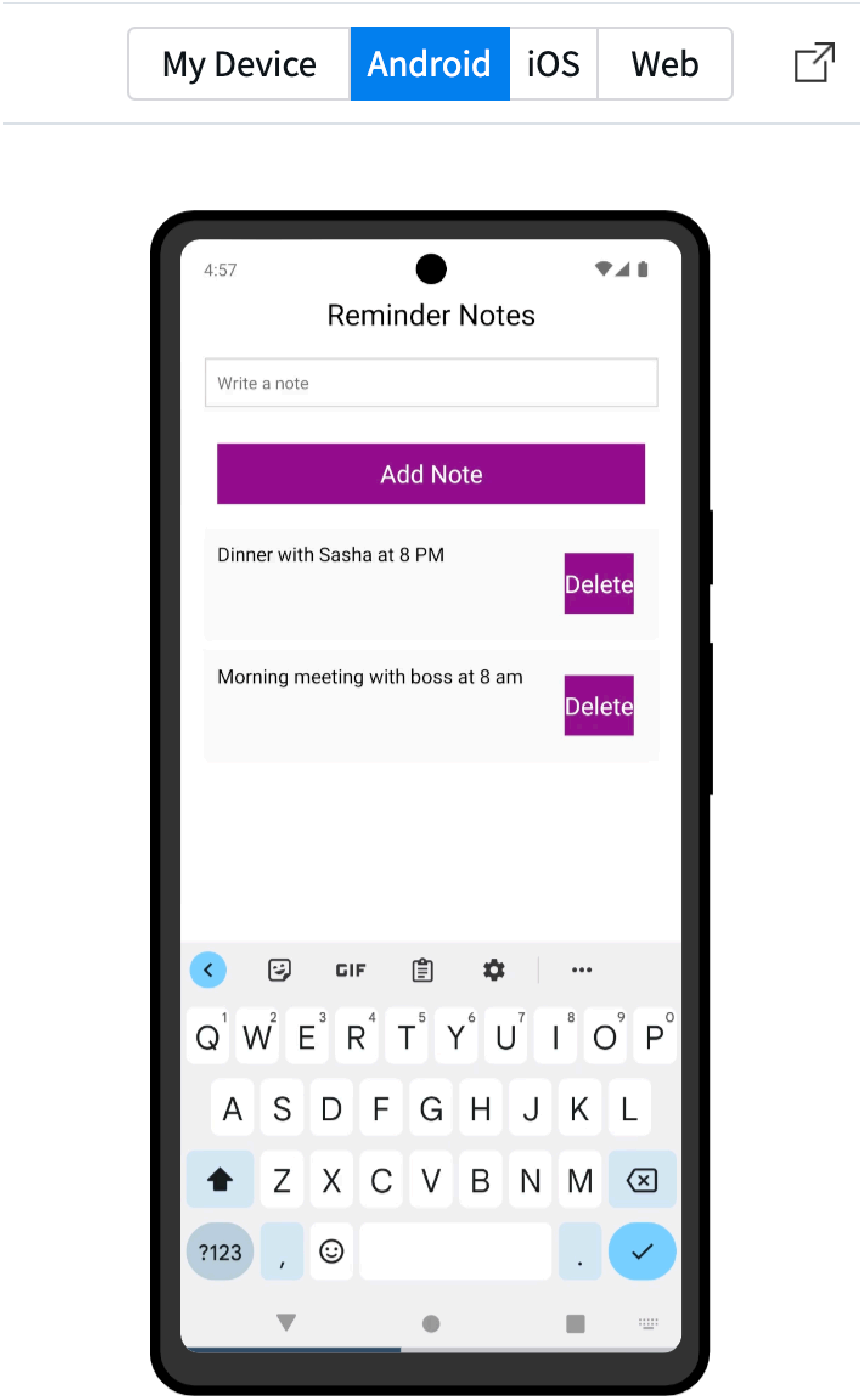
```
    fontSize: 24,
    marginBottom: 20,
    textAlign: 'center',
    marginTop: 25
  },
  input: {
    height: 40,
    borderColor: '#ccc',
    borderWidth: 1,
    marginBottom: 20,
    paddingHorizontal: 10,
  },
  noteContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    padding: 10,
    backgroundColor: '#f9f9f9',
    marginTop: 10,
  },
  noteText: {
    fontSize: 16,
  },
});

export default App;
```

2. Your app will get loaded and you will be able to add notes by entering text either using the keyboard or the device keyboard.

My Device
Android
iOS
Web


4:55
Reminder Notes
Write a note
Add Note

3. Once you add notes you will be able to see them being displayed below. You can click the `delete` button to delete the note.



Reminder Notes

Write a note

Add Note

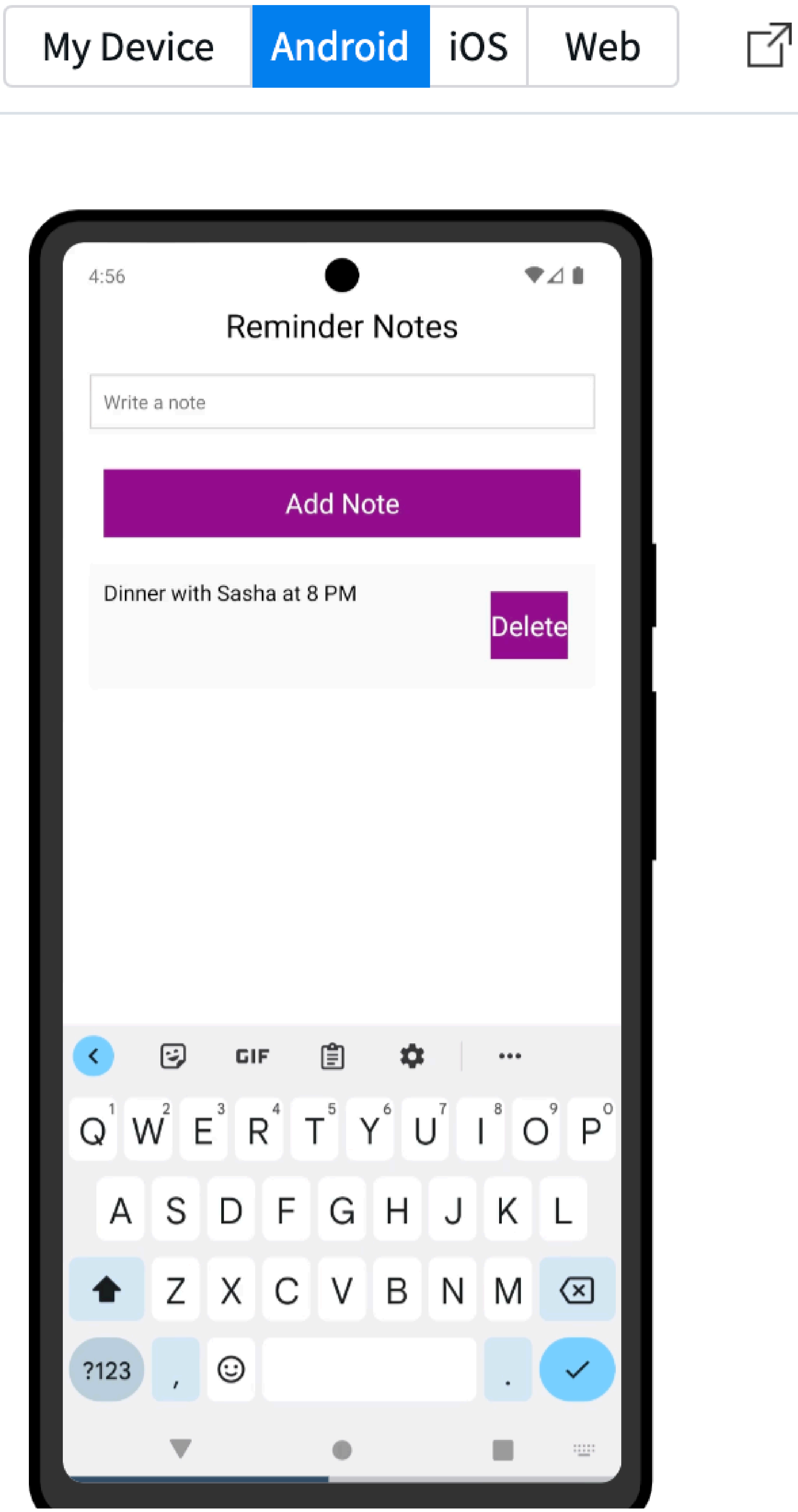
Dinner with Sasha at 8 PM

Delete

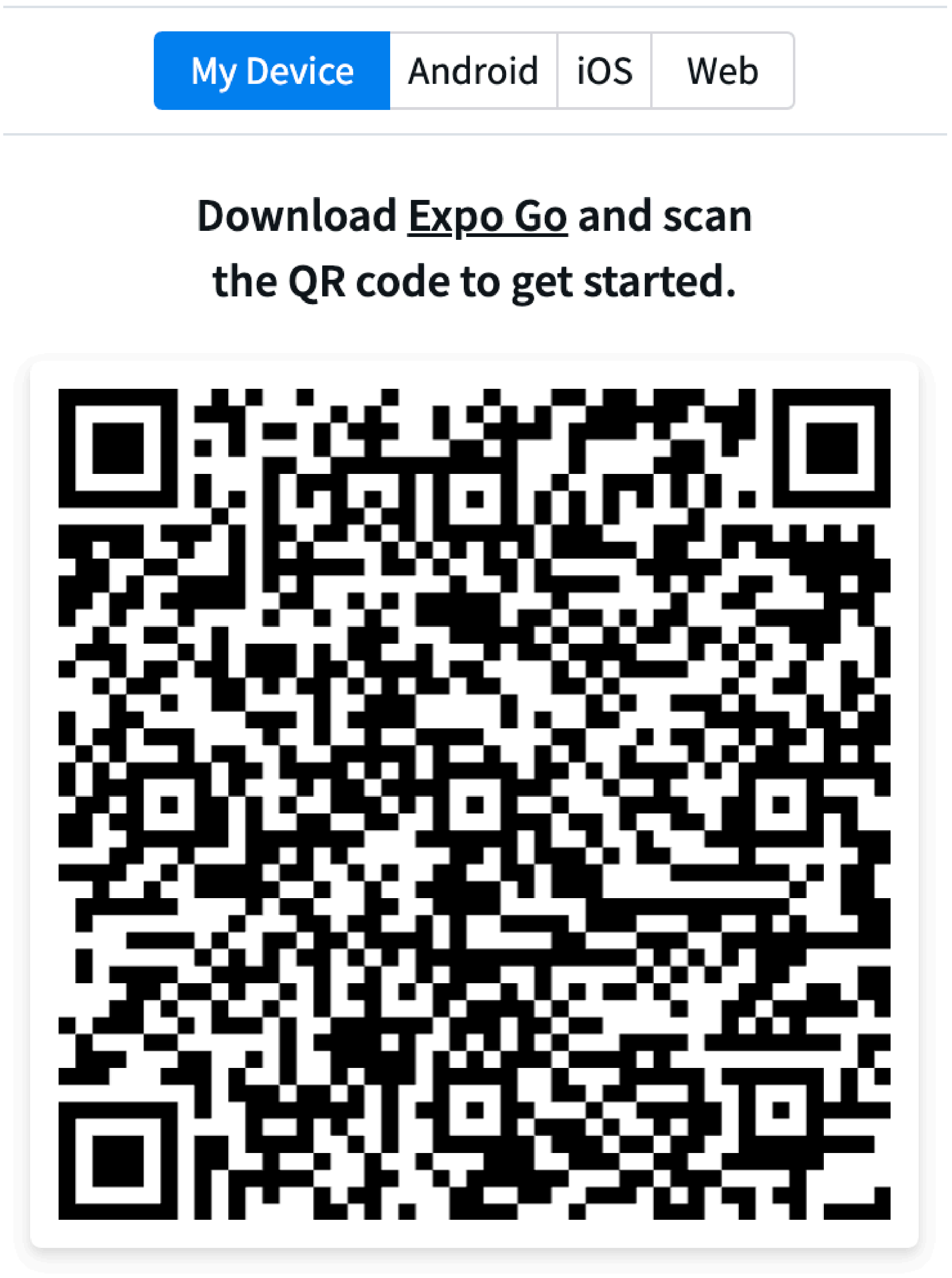
Morning meeting with boss at 8 am

Delete

4. You will see the updated list excluding the note your removed.

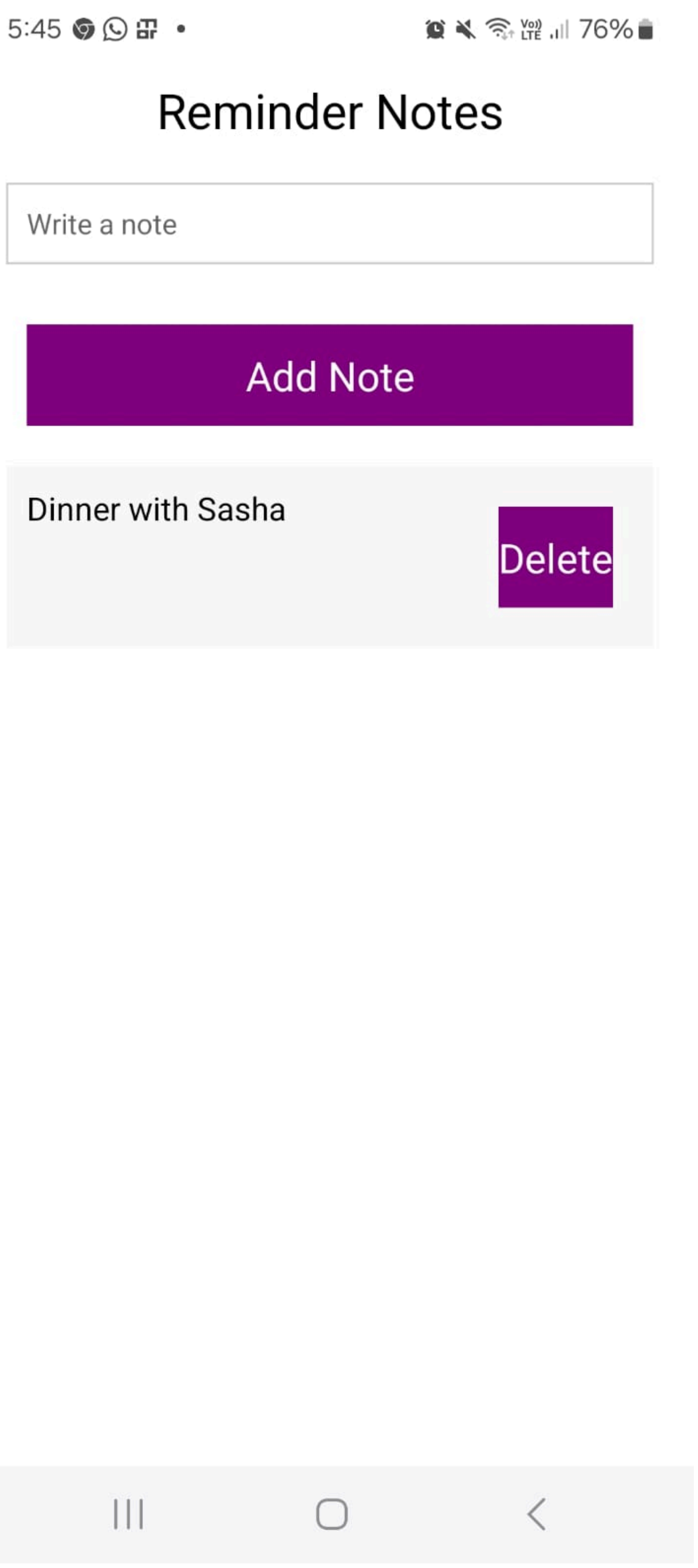

My Device
Android
iOS
Web
4:56
Reminder Notes
Write a note
Add Note
Dinner with Sasha at 8 PM
Delete
GIF
QWERTYUIOP
ASDFGHJKL
ZXCVBNM
?123

5. If you have an Android device, with Expo installed, you can click `My Device` option on the browser and scan QR Code with the `Expo app` to install the app on the phone and test it. This app doesn't work on iPhone as it requires explicit permission to use SQLite on the device.



Download Expo Go and scan the QR code to get started.

6. You can see that the app renders as in the image below on the Android device.


5:45
76%
Reminder Notes
Write a note
Add Note
Dinner with Sasha
Delete

Practice Exercise

1. Change the delete buttons to icons.
2. Add multiple notes and see how the app handles scrolling.

Conclusion

Congratulations on completing this lab! You have now learned how to add expo-sqlite to your React app to handle in-memory storage and create a React Native application that will allow users to store, retrieve, and delete notes. You also know how to manage the database operations from within the React Native app in real time.

Author(s)

Lavanya