# How to make a React-sqlite project for android

**Goal:** Create a react Native app that interfaces with a sqlite database.

**Tools:**

* React, the library
* Vite

**Prerequisites:**

* Know how to create a react native project

#### Installation:

1. Make a new react project
2. Install sqlite into your project
   1. npm install --save react-native-sqlite-storage

#### Create a project with Vite

1. Run this command to create a react project with typescript
   1. *npm create vite@latest app-name –template react-ts*
   2. You might be met with the following menus
      1. A screenshot of a computer program

         AI-generated content may be incorrect.
   * In this menu select react
     1. A screen shot of a computer script

        AI-generated content may be incorrect.
   * In this menu select typescript or typescript + swc
   * Swc is “speedy web compiler” it is an experimental web compiler that claims to be faster.
   1. cd app-demo-1
   2. npm install
   3. npm run dev

#### Code:

1. We will primarily be working in > \app-demo-1\src\App.tsx
   1. App.tsx is essentially your main page that people will land on first
2. Code:

export default App

1. import { useState, useEffect } from 'react';
2. import initSqlJs, { Database, QueryExecResult } from 'sql.js';
3. import './App.css';
4. // what is in this file?
5. //  this is a simple sqlite demonstration for a react frontend
6. //  sqlite is done to store data locally in a database for a front end for like caching data
7. //  or for prototypes and projects that don't need server side databases
8. //  it is used essentially anywhere you need a local db
9. // yea im putting it right in the main app component, usually you would have a whole
10. // ts file dedicated to this interaction layer
11. export default *function* App() {
12. *const* [db, setDb] = useState<Database | *null*>(null); // state management for database object
13. *const* [items, setItems] = useState<*Array*<{ id: *number*; name: *string* }>>([]);
14. *const* [newItem, setNewItem] = useState<*string*>('');
15. // Initialize SQLite
16. // UseEffect is a react hook for running things with dependencies when the page loads after the intial render
17. // in other words its for using non-ts/js code
18. useEffect(() *=>* {
19. *const* initialize = async () *=>* {        //declaring initialize function
20. *const* newDb = await loadFromLocalStorage();     //declaring the db
21. setDb(newDb);                         //state management function, see declaration ^
22. };
23. initialize();
24. }, []);
25. // Query Helper
26. // this function  gets the data from the database
27. *const* query = (*sql*: *string*, *params*: *any*[] = []): QueryExecResult[] *=>* {
28. if (!db) return []; // no database exception
29. try {
30. return db.exec(*sql*, *params*);  //.execute is a sqlite function that will execute sql commands
31. } catch (e) {
32. console.error("Query error:", e);
33. return [];
34. }
35. };
36. // Insert New Item
37. // db.run is just another execution function but nothing is returned
38. *const* addItem = () *=>* {
39. if (!db || !newItem.trim()) return;
40. db.run("INSERT INTO items (name) VALUES (?)", [newItem]);
41. setNewItem('');
42. refreshItems();
43. };
44. // Refresh List, this just pulls all the data from the database to display it
45. //the list,
46. *const* refreshItems = () *=>* {
47. *const* result = query("SELECT \* FROM items");
48. if (result.length > 0) {
49. *const* { columns, values } = result[0];
50. *const* parsed = values.map(*row* *=>* Object.fromEntries(row.map((*val*, *i*) *=>* [columns[i], val]))) as Array<{ id: *number*, name: *string* }>;
51. setItems(parsed);
52. }
53. };
54. // if a database exists, load it
55. useEffect(() *=>* {
56. if (db) refreshItems();
57. }, [db]);
58. // these two functions load the database from actual storage
59. *const* saveToLocalStorage = () *=>* {
60. if (db) {
61. *const* data = db.export(); // Get Uint8Array of the DB
62. *const* base64 = btoa(String.fromCharCode(...data)); // Convert to base64
63. localStorage.setItem('sqlite-db', base64);
64. }
65. };
66. // Utility to load DB from localStorage
67. *const* loadFromLocalStorage = async () *=>* {
68. *const* SQL = await initSqlJs({ locateFile: *file* *=>* `https://sql.js.org/dist/${file}` });
69. *const* saved = localStorage.getItem('sqlite-db');
70. if (saved) {
71. *const* bytes = Uint8Array.from(atob(saved), *c* *=>* c.charCodeAt(0));
72. return new SQL.Database(bytes);
73. } else {
74. *const* db = new SQL.Database();
75. //if we dont have any database, we create the database
76. db.run("CREATE TABLE IF NOT EXISTS items (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT)");
77. return db;
78. }
79. };
80. // reset function for demonstration, just deletes all the data
81. *const* resetDB = () *=>* {
82. localStorage.removeItem('sqlite-db');
83. window.location.reload();
84. };
86. // UI
87. return (
88. <div className="min-h-screen bg-gray-100 p-6 text-gray-800">
89. <div className="max-w-xl mx-auto bg-white rounded-2xl shadow p-6">
90. <h1 className="text-2xl font-bold mb-4">SQLite in React (sql.js)</h1>
91. <div className="flex gap-2 mb-4">
92. <input
93. type="text"
94. placeholder="Enter item name"
95. value={newItem}
96. onChange={*e* *=>* setNewItem(e.target.value)}
97. className="flex-1 border border-gray-300 rounded-xl px-4 py-2"
98. />
99. <button
100. onClick={addItem}
101. className="bg-blue-500 text-white px-4 py-2 rounded-xl hover:bg-blue-600 transition"
102. >
103. Add
104. </button>
105. <button
106. onClick={saveToLocalStorage}
107. className="bg-blue-500 text-white px-4 py-2 rounded-xl hover:bg-blue-600 transition"
108. >
109. save
110. </button>
111. <button
112. onClick={resetDB}
113. className="bg-blue-500 text-white px-4 py-2 rounded-xl hover:bg-blue-600 transition"
114. >
115. reset
116. </button>
117. </div>
118. <ul className="space-y-2">
119. {items.map(*item* *=>* (
120. <li
121. key={item.id}
122. className="bg-blue-100 text-blue-900 px-4 py-2 rounded-xl shadow-sm"
123. >
124. {item.name} <span className="text-sm text-gray-500">(id: {item.id})</span>
125. </li>
126. ))}
127. </ul>
128. </div>
129. </div>
130. );
131. }