# How to make a React-sqlite project

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

**Tools:**

* React, the library
* Vite

**Prerequisites:**

* Know how to create a react project

#### Installation:

1. Make a new react project
2. Install sqlite into your react project
   1. npm install sql.js

#### 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 install sql.js
   4. 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:
3. import { useState, useEffect } from 'react';
4. import initSqlJs, { Database, QueryExecResult } from 'sql.js';
5. import './App.css';
6. // what is in this file?
7. //  this is a simple sqlite demonstration for a react frontend
8. //  sqlite is done to store data locally in a database for a front end for like caching data
9. //  or for prototypes and projects that don't need server side databases
10. //  it is used essentially anywhere you need a local db
11. // yea im putting it right in the main app component, usually you would have a whole
12. // ts file dedicated to this interaction layer
13. export default *function* App() {
14. *const* [db, setDb] = useState<Database | *null*>(null); // state management for database object
15. *const* [items, setItems] = useState<*Array*<{ id: *number*; name: *string* }>>([]);
16. *const* [newItem, setNewItem] = useState<*string*>('');
17. // Initialize SQLite
18. // UseEffect is a react hook for running things with dependencies when the page loads after the intial render
19. // in other words its for using non-ts/js code
20. useEffect(() *=>* {
21. *const* initialize = async () *=>* {        //declaring initialize function
22. *const* newDb = await loadFromLocalStorage();     //declaring the db
23. setDb(newDb);                         //state management function, see declaration ^
24. };
25. initialize();
26. }, []);
27. // Query Helper
28. // this function  gets the data from the database
29. *const* query = (*sql*: *string*, *params*: *any*[] = []): QueryExecResult[] *=>* {
30. if (!db) return []; // no database exception
31. try {
32. return db.exec(*sql*, *params*);  //.execute is a sqlite function that will execute sql commands
33. } catch (e) {
34. console.error("Query error:", e);
35. return [];
36. }
37. };
38. // Insert New Item
39. // db.run is just another execution function but nothing is returned
40. *const* addItem = () *=>* {
41. if (!db || !newItem.trim()) return;
42. db.run("INSERT INTO items (name) VALUES (?)", [newItem]);
43. setNewItem('');
44. refreshItems();
45. };
46. // Refresh List, this just pulls all the data from the database to display it
47. //the list,
48. *const* refreshItems = () *=>* {
49. *const* result = query("SELECT \* FROM items");
50. if (result.length > 0) {
51. *const* { columns, values } = result[0];
52. *const* parsed = values.map(*row* *=>* Object.fromEntries(row.map((*val*, *i*) *=>* [columns[i], val]))) as Array<{ id: *number*, name: *string* }>;
53. setItems(parsed);
54. }
55. };
56. // if a database exists, load it
57. useEffect(() *=>* {
58. if (db) refreshItems();
59. }, [db]);
60. // these two functions load the database from actual storage
61. *const* saveToLocalStorage = () *=>* {
62. if (db) {
63. *const* data = db.export(); // Get Uint8Array of the DB
64. *const* base64 = btoa(String.fromCharCode(...data)); // Convert to base64
65. localStorage.setItem('sqlite-db', base64);
66. }
67. };
68. // Utility to load DB from localStorage
69. *const* loadFromLocalStorage = async () *=>* {
70. *const* SQL = await initSqlJs({ locateFile: *file* *=>* `https://sql.js.org/dist/${file}` });
71. *const* saved = localStorage.getItem('sqlite-db');
72. if (saved) {
73. *const* bytes = Uint8Array.from(atob(saved), *c* *=>* c.charCodeAt(0));
74. return new SQL.Database(bytes);
75. } else {
76. *const* db = new SQL.Database();
77. //if we dont have any database, we create the database
78. db.run("CREATE TABLE IF NOT EXISTS items (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT)");
79. return db;
80. }
81. };
82. // reset function for demonstration, just deletes all the data
83. *const* resetDB = () *=>* {
84. localStorage.removeItem('sqlite-db');
85. window.location.reload();
86. };
88. // UI
89. return (
90. <div className="min-h-screen bg-gray-100 p-6 text-gray-800">
91. <div className="max-w-xl mx-auto bg-white rounded-2xl shadow p-6">
92. <h1 className="text-2xl font-bold mb-4">SQLite in React (sql.js)</h1>
93. <div className="flex gap-2 mb-4">
94. <input
95. type="text"
96. placeholder="Enter item name"
97. value={newItem}
98. onChange={*e* *=>* setNewItem(e.target.value)}
99. className="flex-1 border border-gray-300 rounded-xl px-4 py-2"
100. />
101. <button
102. onClick={addItem}
103. className="bg-blue-500 text-white px-4 py-2 rounded-xl hover:bg-blue-600 transition"
104. >
105. Add
106. </button>
107. <button
108. onClick={saveToLocalStorage}
109. className="bg-blue-500 text-white px-4 py-2 rounded-xl hover:bg-blue-600 transition"
110. >
111. save
112. </button>
113. <button
114. onClick={resetDB}
115. className="bg-blue-500 text-white px-4 py-2 rounded-xl hover:bg-blue-600 transition"
116. >
117. reset
118. </button>
119. </div>
120. <ul className="space-y-2">
121. {items.map(*item* *=>* (
122. <li
123. key={item.id}
124. className="bg-blue-100 text-blue-900 px-4 py-2 rounded-xl shadow-sm"
125. >
126. {item.name} <span className="text-sm text-gray-500">(id: {item.id})</span>
127. </li>
128. ))}
129. </ul>
130. </div>
131. </div>
132. );
133. }