Below is a minimal “getting-started” code scaffold in plain JavaScript/JSX (no TypeScript) for a Vite + React + Tailwind flashcard app. It includes:

1. **Project setup notes**
2. **Folder structure**
3. **Key files with boilerplate code**:
   - `useLocalStorage` hook
   - `AppContext` (global state)
   - `main.jsx` & `App.jsx` (router setup)
   - A simple **Home** page (folder/deck CRUD)
   - An **ImportDeck** component (CSV → deck)
   - A bare-bones **Study** page & **FlashcardViewer** component

You can copy/paste these into your `src/` folder and adjust as you build out features.

---

1. Project Setup

1. **Create a new Vite + React + Tailwind project** (if you haven’t already):

```
npm create vite@latest my-flashcards-app -- --template react
cd my-flashcards-app
npm install
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

2. **Configure Tailwind** in `tailwind.config.js` (auto-generated by `npx tailwindcss init -p`):

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    "./index.html",
    "./src/**/*.{js,jsx}"
  ],
  theme: {
    extend: {}
  },
  plugins: []
}
```

3. **In `src/index.css`**, add the Tailwind directives:

```
@tailwind base;
@tailwind components;
```

```
@tailwind utilities;
```

4. **Ensure your `vite.config.js`** has no special React/Tailwind settings beyond the default. Vite's React template already handles JSX, so you shouldn't need to touch it for now.

5. **Install React Router** (we'll want at least a "Home" and "Study" route):

```
npm install react-router-dom
```

---

2. Folder Structure

In `src/`, create these folders/files:

```
src/
├─ main.jsx
├─ App.jsx
├─ index.css
│
├─ hooks/
│  └─ useLocalStorage.js
│
├─ contexts/
│  └─ AppContext.jsx
│
├─ utils/
│  ├─ shuffle.js
│  └─ csvParser.js
│
├─ components/
│  ├─ FolderTree.jsx
│  ├─ DeckList.jsx
│  ├─ ImportDeck.jsx
│  └─ FlashcardViewer.jsx
│
├─ pages/
│  ├─ Home.jsx
│  └─ Study.jsx
└─ (any other assets or helpers)
```

---

3. `src/hooks/useLocalStorage.js`

A simple React hook to sync state ↔ `localStorage`.

```
// src/hooks/useLocalStorage.js
import { useState, useEffect } from "react";
```

```
/**
 * Hook: useLocalStorage
 *
 * @param {string} key          The localStorage key to read/write.
 * @param {any} defaultValue    The default value (if nothing is in
localStorage).
 * @returns {[state, setState]} A state value and setter; writing updates
localStorage.
 */
export function useLocalStorage(key, defaultValue) {
  const [state, setState] = useState(() => {
    try {
      const stored = localStorage.getItem(key);
      return stored !== null
        ? JSON.parse(stored)
        : defaultValue;
    } catch (e) {
      console.error("useLocalStorage parse error:", e);
      return defaultValue;
    }
  });

  useEffect(() => {
    try {
      localStorage.setItem(key, JSON.stringify(state));
    } catch (e) {
      console.error("useLocalStorage write error:", e);
    }
  }, [key, state]);

  return [state, setState];
}
```

---

4. `src/contexts/AppContext.jsx`

Global state containing arrays of folders and decks. We'll store it under one `localStorage` key so it persists across reloads.

```
// src/contexts/AppContext.jsx
import React, { createContext, useContext } from "react";
import { useLocalStorage } from "../hooks/useLocalStorage";

/**
 * A “Folder” object:
 *   { id, name, parentId }
 *
 * A “Deck” object:
 *   { id, name, folderId, cards: [ { id, front, back } ] }
 *
```

```
 * AppState shape:
 *   {
 *     folders: Folder[],
 *     decks: Deck[],
 *     selectedFolderId: string|null
 *   }
 */
const defaultAppState = {
  folders: [],
  decks: [],
  selectedFolderId: null
};

// Create context
const AppContext = createContext(null);

/**
 * AppProvider wraps <App/> and provides global state via context.
 */
export function AppProvider({ children }) {
  // Use localStorage for persisting state
  const [state, setState] = useLocalStorage(
    "flashcards_app_state",
    defaultAppState
  );

  return (
    <AppContext.Provider value={{ state, setState }}>
      {children}
    </AppContext.Provider>
  );
}

/**
 * Custom hook to read/write AppContext.
 */
export function useAppContext() {
  const ctx = useContext(AppContext);
  if (!ctx) {
    throw new Error("useAppContext must be used inside AppProvider");
  }
  return ctx;
}
```

---

5. src/utils/shuffle.js

A Fisher–Yates shuffle for an array of IDs.

```
// src/utils/shuffle.js
```

```
/**
 * Shuffle an array in place (Fisher–Yates).
 * Returns the same array reference (shuffled).
 */
export function shuffle(array) {
  let m = array.length, i;
  while (m) {
    i = Math.floor(Math.random() * m--);
    [array[m], array[i]] = [array[i], array[m]];
  }
  return array;
}
```

---

6. src/utils/csvParser.js

A minimal CSV parser: split lines, split on comma. (You can swap in PapaParse later if you need robust parsing.)

```
// src/utils/csvParser.js

/**
 * parseCsv(text)
 *   Splits by line breaks, then by first comma.
 *   Returns an array of { front, back } objects.
 *
 * Assumes CSV format: each line = "front,back"
 */
export function parseCsv(text) {
  return text
    .trim()
    .split("\n")
    .map((line) => {
      const [front, back] = line.split(",");
      return {
        front: front ? front.trim() : "",
        back: back ? back.trim() : ""
      };
    })
    .filter(({ front, back }) => front !== "" && back !== "");
}
```

---

7. src/main.jsx

Entry point: wrap <App/> with <AppProvider>, import Tailwind's styles, and mount.

```
// src/main.jsx
import React from "react";
```

```
import ReactDOM from "react-dom/client";
import App from "./App";
import { AppProvider } from "./contexts/AppContext";
import "./index.css"; // Tailwind directives

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <AppProvider>
      <App />
    </AppProvider>
  </React.StrictMode>
);
```

---

8. src/App.jsx

Sets up React Router with two routes: “Home” and “Study”. You can add more later (e.g. a Settings page).

```
// src/App.jsx
import React from "react";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Home from "./pages/Home";
import Study from "./pages/Study";

export default function App() {
  return (
    <BrowserRouter>
      <div className="min-h-screen bg-gray-50 text-gray-800">
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/study/deck/:deckId" element={<Study />} />
          <Route path="/study/folder/:folderId" element={<Study />} />
        </Routes>
      </div>
    </BrowserRouter>
  );
}
```

---

9. src/pages/Home.jsx

“Home” is where you:

- Display the nested folder tree
- Display decks within the selected folder
- Provide buttons/UI to add/rename/delete folders and decks
- Allow importing a new deck via CSV

Home.jsx pulls from `AppContext` and renders `FolderTree`, `DeckList`, and an “ImportDeck” button/modal.

```
// src/pages/Home.jsx
import React, { useState, useMemo } from "react";
import { useAppContext } from "../contexts/AppContext";
import { FolderTree } from "../components/FolderTree";
import { DeckList } from "../components/DeckList";
import ImportDeck from "../components/ImportDeck";

export default function Home() {
  const { state, setState } = useAppContext();
  const { folders, decks, selectedFolderId } = state;
  const [importingForFolder, setImportingForFolder] = useState(null);

  // Compute decks that belong to the currently selected folder
  const decksInFolder = useMemo(() => {
    return decks.filter((d) => d.folderId === selectedFolderId);
  }, [decks, selectedFolderId]);

  return (
    <div className="flex h-screen">
      {/* Sidebar: nested folder tree */}
      <aside className="w-64 border-r bg-white p-4 overflow-y-auto">
        <h2 className="text-xl font-semibold mb-4">Folders</h2>
        <FolderTree />
        {/* Button to add a new top-level folder */}
        <button
          className="mt-4 w-full py-2 bg-blue-500 text-white rounded
hover:bg-blue-600"
          onClick={() => {
            const name = prompt("New folder name:");
            if (name) {
              const newFolder = {
                id: crypto.randomUUID(),
                name,
                parentId: null
              };
              setState((prev) => ({
                ...prev,
                folders: [...prev.folders, newFolder]
              }));
            }
          }}
        >
          + New Folder
        </button>
      </aside>

      {/* Main area: deck list + import button */}
      <main className="flex-1 p-6 overflow-y-auto">
        <h2 className="text-2xl font-semibold mb-4">
```

```
          {selectedFolderId
            ? `Decks in “${folders.find(f => f.id ===
selectedFolderId)?.name
            }”`
            : "Select a folder"}
        </h2>

        {selectedFolderId ? (
          <>
            <DeckList decks={decksInFolder} />
            <button
              className="mt-4 py-2 px-4 bg-green-500 text-white rounded
hover:bg-green-600"
              onClick={() => setImportingForFolder(selectedFolderId)}
            >
              + Import Deck from CSV
            </button>
          </>
        ) : (
          <p className="text-gray-600">No folder selected.</p>
        )}

        {/* ImportDeck modal/dialog */}
        {importingForFolder && (
          <ImportDeck
            folderId={importingForFolder}
            onClose={() => setImportingForFolder(null)}
          />
        )}
      </main>
    </div>
  );
}
```

9.1. src/components/FolderTree.jsx

A recursive tree view of folders. Clicking a folder sets it as “selected” in context.

```
// src/components/FolderTree.jsx
import React from "react";
import { useAppContext } from "../contexts/AppContext";

/**
 * Recursively render all child folders whose parentId === provided
parentId.
 */
function FolderNode({ folder, level = 0 }) {
  const { state, setState } = useAppContext();
  const { folders, selectedFolderId } = state;
```

```
    // Find direct children of this folder
    const children = folders.filter((f) => f.parentId === folder.id);

    const isSelected = selectedFolderId === folder.id;

    return (
      <div className={`pl-${level * 4} mb-2`}>
        <div className="flex items-center space-x-2">
          <span
            className={`cursor-pointer ${
              isSelected ? "font-bold text-blue-600" : "text-gray-800"
            }`}
            onClick={() =>
              setState((prev) => ({ ...prev, selectedFolderId: folder.id }))
            }
          >
            {folder.name}
          </span>

          {/* Rename button */}
          <button
            className="text-sm text-gray-500 hover:text-gray-800"
            onClick={() => {
              const newName = prompt("Rename folder:", folder.name);
              if (newName) {
                setState((prev) => ({
                  ...prev,
                  folders: prev.folders.map((f) =>
                    f.id === folder.id ? { ...f, name: newName } : f
                  )
                }));
              }
            }}
          >
            ✎
          </button>

          {/* Delete button (cascades deletes child folders & decks) */}
          <button
            className="text-sm text-red-500 hover:text-red-700"
            onClick={() => {
              if (
                confirm(
                  `Delete folder "${folder.name}" and all its contents?`
                )
              ) {
                // Compute all descendant folder IDs
                const gatherDescendants = (id, allFolders) => {
                  let out = [id];
                  allFolders
                    .filter((f) => f.parentId === id)
                    .forEach((ch) => {
                      out = out.concat(gatherDescendants(ch.id, allFolders));
                    });
```

```
                return out;
              };
              const toDeleteIds = gatherDescendants(folder.id, folders);

              setState((prev) => ({
                ...prev,
                // remove folders whose id is in toDeleteIds
                folders: prev.folders.filter(
                  (f) => !toDeleteIds.includes(f.id)
                ),
                // remove decks in those folders
                decks: prev.decks.filter(
                  (d) => !toDeleteIds.includes(d.folderId)
                ),
                // if selectedFolderId was under those, clear it
                selectedFolderId:
                  toDeleteIds.includes(prev.selectedFolderId)
                    ? null
                    : prev.selectedFolderId
              }));
            }
          }}
        >
          🗑
        </button>
      </div>

      {/* Render its children */}
      {children.map((child) => (
        <FolderNode key={child.id} folder={child} level={level + 1} />
      ))}
    </div>
  );
}

export function FolderTree() {
  const { state } = useAppContext();
  // Top-level folders have parentId === null
  const topFolders = state.folders.filter((f) => f.parentId === null);

  return (
    <div>
      {topFolders.map((folder) => (
        <FolderNode key={folder.id} folder={folder} level={0} />
      ))}
    </div>
  );
}
```

9.2. src/components/DeckList.jsx

Lists all decks in the current folder. Includes “Study” and “Delete” buttons.

```
// src/components/DeckList.jsx
import React from "react";
import { Link } from "react-router-dom";
import { useAppContext } from "../contexts/AppContext";

export function DeckList({ decks }) {
  const { setState } = useAppContext();

  if (decks.length === 0) {
    return <p className="text-gray-600">No decks in this folder.</p>;
  }

  return (
    <ul className="space-y-2">
      {decks.map((deck) => (
        <li
          key={deck.id}
          className="flex items-center justify-between bg-white p-3 rounded
shadow-sm"
        >
          <span className="font-medium">{deck.name}</span>
          <div className="space-x-2">
            {/* Study button: navigates to /study/deck/:deckId */}
            <Link to={`/study/deck/${deck.id}`}>
              <button className="px-3 py-1 bg-blue-500 text-white rounded
hover:bg-blue-600">
                Study
              </button>
            </Link>

            {/* Delete deck */}
            <button
              className="px-2 py-1 bg-red-500 text-white rounded hover:bg-
red-600"
              onClick={() => {
                if (confirm(`Delete deck "${deck.name}"?`)) {
                  setState((prev) => ({
                    ...prev,
                    decks: prev.decks.filter((d) => d.id !== deck.id)
                  }));
                }
              }}
            >
              🗑
            </button>
          </div>
        </li>
      ))}
    </ul>
  );
}
```

9.3. src/components/ImportDeck.jsx

A modal/dialog that lets the user upload a CSV file. Once parsed, it creates a new Deck object in state and closes.

```
// src/components/ImportDeck.jsx
import React, { useState } from "react";
import { useAppContext } from "../contexts/AppContext";
import { parseCsv } from "../utils/csvParser";

export default function ImportDeck({ folderId, onClose }) {
  const { setState } = useAppContext();
  const [fileError, setFileError] = useState("");

  const handleFile = (event) => {
    const file = event.target.files[0];
    if (!file) return;

    if (!file.name.endsWith(".csv")) {
      setFileError("Please upload a CSV file.");
      return;
    }

    const reader = new FileReader();
    reader.onload = () => {
      try {
        const text = reader.result;
        const parsed = parseCsv(text);

        if (parsed.length === 0) {
          setFileError("CSV is empty or not formatted properly.");
          return;
        }

        // Build new deck object
        const newDeck = {
          id: crypto.randomUUID(),
          name: prompt("Enter a name for this deck:", "Imported Deck") ||
"Imported Deck",
          folderId: folderId,
          cards: parsed.map(({ front, back }) => ({
            id: crypto.randomUUID(),
            front,
            back
          }))
        };

        // Save to global state
        setState((prev) => ({
```

```
          ...prev,
          decks: [...prev.decks, newDeck]
        }));

        onClose();
      } catch (err) {
        console.error(err);
        setFileError("Error parsing CSV.");
      }
    };
    reader.readAsText(file);
  };

  return (
    <div className="fixed inset-0 bg-black bg-opacity-40 flex items-center
justify-center z-50">
      <div className="bg-white p-6 rounded-lg shadow-lg w-96">
        <h3 className="text-xl font-semibold mb-4">Import Deck</h3>
        <input type="file" accept=".csv" onChange={handleFile} />
        {fileError && <p className="text-red-500 mt-2">{fileError}</p>}
        <div className="mt-4 flex justify-end space-x-2">
          <button
            className="px-4 py-2 bg-gray-300 rounded hover:bg-gray-400"
            onClick={onClose}
          >
            Cancel
          </button>
        </div>
      </div>
    </div>
  );
}
```

---

10. src/pages/Study.jsx

When the user clicks “Study” on a deck (or a folder), we show flashcards one by one, allow “Right/Wrong,” and track progress in `localStorage`.

```
// src/pages/Study.jsx
import React, { useMemo, useEffect, useState } from "react";
import { useParams, Link } from "react-router-dom";
import { useAppContext } from "../contexts/AppContext";
import { shuffle } from "../utils/shuffle";
import FlashcardViewer from "../components/FlashcardViewer";

export default function Study() {
  const { deckId, folderId } = useParams();
  const { state, setState } = useAppContext();
  const { decks, folders } = state;
```

```
  // Determine which cards to study:
  const cardsToStudy = useMemo(() => {
    if (deckId) {
      const deck = decks.find((d) => d.id === deckId);
      return deck ? [...deck.cards] : [];
    }
    if (folderId) {
      // Recursively gather all cards under this folder
      const gatherDescendants = (id, allFolders) => {
        let out = [id];
        allFolders
          .filter((f) => f.parentId === id)
          .forEach((ch) => {
            out = out.concat(gatherDescendants(ch.id, allFolders));
          });
        return out;
      };
      const folderIds = gatherDescendants(folderId, folders);
      return decks
        .filter((d) => folderIds.includes(d.folderId))
        .flatMap((d) => d.cards);
    }
    return [];
  }, [deckId, folderId, decks, folders]);

  // Session state key (string) in localStorage:
  const sessionKey = deckId ? `session_deck_${deckId}` :
`session_folder_${folderId}`;

  // Load or initialize session from localStorage
  const [session, setSession] = useState(() => {
    try {
      const stored = localStorage.getItem(sessionKey);
      if (stored) {
        // parse and convert incorrectIds back to a Set
        const parsed = JSON.parse(stored);
        return {
          cardOrder: parsed.cardOrder,
          currentIndex: parsed.currentIndex,
          incorrectIds: new Set(parsed.incorrectIds)
        };
      }
    } catch (e) {
      console.error("Session parse error:", e);
    }
    // If no session in storage, initialize:
    const ids = cardsToStudy.map((c) => c.id);
    return {
      cardOrder: shuffle(ids.slice()), // shuffle a copy
      currentIndex: 0,
      incorrectIds: new Set()
    };
  });
```

```
    // Whenever session changes, persist to localStorage
    useEffect(() => {
      const toStore = {
        cardOrder: session.cardOrder,
        currentIndex: session.currentIndex,
        incorrectIds: Array.from(session.incorrectIds)
      };
      localStorage.setItem(sessionKey, JSON.stringify(toStore));
    }, [session, sessionKey]);

    // If cardsToStudy changes (e.g. new import), reset session
    useEffect(() => {
      const ids = cardsToStudy.map((c) => c.id);
      setSession({
        cardOrder: shuffle(ids.slice()),
        currentIndex: 0,
        incorrectIds: new Set()
      });
    }, [cardsToStudy]);

    if (cardsToStudy.length === 0) {
      return (
        <div className="p-6">
          <p className="text-gray-600">
            {deckId
              ? "Deck is empty or not found."
              : "No cards found under this folder."}
          </p>
          <Link to="/" className="text-blue-500 hover:underline">
            ← Back to Home
          </Link>
        </div>
      );
    }

    const { cardOrder, currentIndex, incorrectIds } = session;

    // If we haven't run out of cards yet:
    if (currentIndex < cardOrder.length) {
      const currentCardId = cardOrder[currentIndex];
      const currentCard = cardsToStudy.find((c) => c.id === currentCardId);

      const markAnswer = (isCorrect) => {
        setSession((prev) => {
          const newIncorrect = new Set(prev.incorrectIds);
          if (!isCorrect && currentCardId) {
            newIncorrect.add(currentCardId);
          }
          return {
            cardOrder: prev.cardOrder,
            currentIndex: prev.currentIndex + 1,
            incorrectIds: newIncorrect
          };
        });
```

```
    };

    return (
      <div className="p-6 flex flex-col items-center">
        <p className="mb-2 text-gray-500">
          Card {currentIndex + 1} of {cardOrder.length}
        </p>
        <FlashcardViewer
          card={currentCard}
          onCorrect={() => markAnswer(true)}
          onWrong={() => markAnswer(false)}
        />
      </div>
    );
  }

  // Otherwise, we've finished all cards: show summary + "Review Wrong" or
"Restart"
  const wrongIdsArray = Array.from(incorrectIds);
  const wrongCards = cardsToStudy.filter((c) =>
wrongIdsArray.includes(c.id));

  return (
    <div className="p-6">
      <h2 className="text-2xl font-semibold mb-4">Session Complete!</h2>
      <p className="mb-4">
        You got {wrongCards.length} out of {cardsToStudy.length} wrong.
      </p>

      {wrongCards.length > 0 && (
        <button
          className="mb-4 px-4 py-2 bg-yellow-500 text-white rounded
hover:bg-yellow-600"
          onClick={() => {
            // Start a mini-session just for wrong cards
            const ids = wrongCards.map((c) => c.id);
            setSession({
              cardOrder: shuffle(ids.slice()),
              currentIndex: 0,
              incorrectIds: new Set()
            });
          }}
        >
          Review Wrong Cards
        </button>
      )}

      <button
        className="mb-4 px-4 py-2 bg-blue-500 text-white rounded hover:bg-
blue-600"
        onClick={() => {
          // Restart full session
          const allIds = cardsToStudy.map((c) => c.id);
          setSession({
```

```
            cardOrder: shuffle(allIds.slice()),
            currentIndex: 0,
            incorrectIds: new Set()
          });
        }}
      >
        Restart Full Session
      </button>

      <br />
      <Link to="/" className="text-blue-500 hover:underline">
        ← Back to Home
      </Link>
    </div>
  );
}
```

10.1. src/components/FlashcardViewer.jsx

A simple card that flips front/back on click, and shows “Right/ Wrong” buttons.

```
// src/components/FlashcardViewer.jsx
import React, { useState } from "react";

export default function FlashcardViewer({ card, onCorrect, onWrong }) {
  const [showBack, setShowBack] = useState(false);

  return (
    <div className="flex flex-col items-center space-y-4">
      <div
        className="w-full max-w-md p-6 bg-white rounded-lg shadow-lg text-
center cursor-pointer select-none"
        onClick={() => setShowBack((prev) => !prev)}
      >
        {showBack ? card.back : card.front}
      </div>
      <div className="flex space-x-4">
        <button
          className="px-4 py-2 bg-green-500 text-white rounded hover:bg-
green-600"
          onClick={onCorrect}
        >
          ✓ Right
        </button>
        <button
          className="px-4 py-2 bg-red-500 text-white rounded hover:bg-red-
600"
          onClick={onWrong}
        >
          ✕ Wrong
```

```
          </button>
        </div>
      </div>
    );
  }
```

---

11. Wrapping Up

Now you have the bare-bones pieces in place:

1. `useLocalStorage` for persisting state
2. `AppContext` for global arrays of folders & decks
3. `Home.jsx` + `FolderTree.jsx` + `DeckList.jsx` + `ImportDeck.jsx` to create, delete, and import.
4. `Study.jsx` + `FlashcardViewer.jsx` to run a study session, shuffle cards, track right/wrong, and let the user review mistakes.

Next steps you might add:

- **"Add Deck by Hand"** (instead of just CSV)
- **Rename/Delete individual cards** inside a deck
- **Toggle "front first" or "back first"** in a small settings modal
- **Export a deck as CSV** (e.g. build a Blob & `a.download`)
- **Dark mode toggle** (store preference in localStorage)
- **Better error handling/UI polish** (tooltips, empty states, etc.)
- **Drag-and-drop reordering** for cards in a deck (e.g. using `react-beautiful-dnd`)
- **IndexedDB** (via `localForage`) if you need to store hundreds of decks/cards

But at this point, you can run:

```
npm run dev
```

…click a folder on the left, import a `.csv` with lines like:

```
Capital of France,Paris
React hook for state,useState
```

…and then hit **Study**, flip cards, mark ✓/✗, and see it remember your progress in `localStorage`.

From here on, simply iterate: style with Tailwind (or ShadCN components), add small features, and enjoy building your frontend-only flashcard app.