Vapor Vault: Offline Password Manager

Concept: Password manager that uses AES-256 encryption and eliminates the server as a point of failure. The manager uses the browser's own engine as an operating environment, and data never leaves the user's device. The entire app compiles in a single, portable HTML file that anyone can run.

Caveat: If the browser data is cleared prior to downloading a backup, all data will be lost. On the other hand, if the user forgets their master password, their data is permanently locked; they must reset the browser data to wipe the vault.

1. Startup:

When the file is first opened, the program checks in the browser's storage for an existing vault with `localStorage.getItem("vaporVault")`.

- If `State.hasVault` is false, it renders the "Create Master Password" screen.
- The password must be at least 8 characters in length to be initialized.
- If the data exists, the app renders the "Enter Password" login screen.

2. Authentication Workflow:

- Key Derivation (PBKDF2):
    1. The app takes the text password
    2. If it is an existing vault, it extracts the Salt from the stored JSON object.
    3. If it is a new vault, it generates a fresh random Salt.
    4. It runs the password and salt through PBKDF2 (100,000 iterations) to generate a cryptokey.
- Decryption:

1. The app decrypts the ciphertext from `localStorage` using AES-GCM (Galois/Counter Mode) and the generated key.
2. Success: The decrypted bytes are decoded into a JSON string, parsed into `State.vault`, and the UI switches to the Dashboard.
3. Failure: `subtle.decrypt` function throws an error and the app displays "ACCESS DENIED".

3. Encryption:

Anytime data is added, edited, or deleted, the `saveVault()` function is triggered.

1. `State.vault` array is updated in memory
2. The array is converted to a string via `JSON.stringify()`.
3. A new 16 byte Salt and 12 byte IV (Initialization Vector) is generated.
4. Re-encryption: The master password is used to derive a new key using the new Salt.
5. The app bundles the Salt, IV, and Ciphertext (vault data) into an object and saves it to `localStorage`, overwriting the previous version.

4. Dashboard Operations:

The user interacts with the `State.vault` array in memory.

- Search: the `renderDashboard()` function filters the array based on `State.search`.
- Create/Edit Entry:
    1. `State.editMode` is set to `'new'` or a specific entry ID.
    2. As the user types, `updateForm()` updates the temporary `State.formData` object.
    3. When "Save" is clicked, the app merges `State.formData` into `State.vault` and encrypts the data.

4. Delete: The app filters the target ID out of the `State.vault` array and data is encrypted again.

5. Advanced Security Features:

- Strength Checker: (1 point per criteria)
    1. Length over 8 characters
    2. Length over 12 characters
    3. Contains an uppercase letter
    4. Contains a number
    5. Contains a special character/symbol
    6. Visual Feedback:
        - 0-2 Points: “WEAK” (Red)
        - 3-4 Points: “MODERATE” (Yellow)
        - 5 Points: “MAXIMUM” (Green)
- Password Generator:
    1. `window.crypto.getRandomValues()` generates 16 random 32-bit integers
    2. It loops through the array of random integers and uses the modulo operator **(`%`)** to map each large random number to a valid index within the character set (0–69).
    3. This produces a 16-character string composed of mixed-case letters, numbers, and symbols.
- Key Rotation:
    1. The user must enter the current master password

2. Current decrypted data is taken, the app triggers `CRYPTO.encrypt` to derive a new key, overwrites `localStorage`, and `State.masterPass` is updated to the new password.

- Backup/Restore:
    1. Backup: The raw encrypted string from `localStorage` is converted to a Blob and triggers a browser download (.json file). This is fully encrypted; if the user loses their password, the backup is useless.
    2. Restore: The app reads the JSON file and then dumps it into `localStorage`, overwriting the current database.
- Nuke (Wipe Data)
    1. The user must enter the master password for confirmation.
    2. `localStorage.removeItem("vaporVault")` is triggered, reloads the page, and returns to “Initialization Phase”.