Building a Persistent Notes App

Using HTML Forms, Web Storage, and AI-Assisted Development

1. Overview of the Program

This project is a **single-page web-based notes application** built using only:

- HTML (for structure)
- CSS (for layout and visual design)
- JavaScript (for behavior and data storage)

The program allows a user to:

- Create notes with a **title** and **body**
- View previously saved notes
- Edit and retitle notes
- Delete notes (with undo support)
- Sort notes alphabetically or by creation date
- Keep all notes saved persistently in the browser

No servers, databases, or external libraries are required.
Everything runs locally inside the user's browser.

---

2. Why This Program Is Educationally Useful

This project demonstrates several important web development concepts:

- How **HTML forms** collect user input
- How a browser can **store data persistently** using localStorage
- How JavaScript connects the user interface to stored data
- How UI design affects behavior (e.g., Save vs. Save As)
- How **AI tools** can be used iteratively to design and improve software

Students can understand how modern apps work **without needing frameworks** or backend systems.

---

3. HTML Forms: How User Input Is Collected

The core of the program is an HTML form-like interface, consisting of:

Title Input

- Used to store the note's title
- Appears in the notes list
- Can be edited and retitled

Note Body

- Used for the note's text
- Has word wrapping
- Has a fixed height with a vertical scrollbar

Notes List

- Displays saved note titles
- Only one note can be selected at a time
- Selecting a note loads it into the editor

These inputs do not automatically save data.
JavaScript is used to read values from them and store them.

---

4. How Notes Are Stored: Web Storage (localStorage)

What Is Web Storage?

Web Storage is a browser feature that allows data to be stored **persistently** in key-value pairs.

This app uses:

```
localStorage
```

Key characteristics:

- Data survives page reloads
- Data is stored per browser per site
- Data is local to the user's device
- No internet connection is required

Notes Data Structure

Each note is stored as a JavaScript object:

```
{
  title: "Example Note",
  body: "This is the note text",
```

```
    created: 1714250000000
}
```

All notes are stored in an array:

```
[
  { title, body, created },
  { title, body, created }
]
```

This array is converted to JSON and saved:

```
localStorage.setItem("notesAppNotes", JSON.stringify(notes));
```

When the page loads, the app restores data using:

```
JSON.parse(localStorage.getItem("notesAppNotes")) || [];
```

This is how the app “remembers” notes.

---

5. Save Logic: Preventing Accidental Data Loss

The program uses **two save buttons**, similar to desktop applications like Word.

Save Note

- If a note is selected → update it
- If no note is selected → create a new one

This prevents the common beginner problem where clicking “Save” does nothing.

Save As New

- Always creates a new note
- Even if an existing note is selected

This clearly separates **editing** from **creating**, making user behavior predictable.

---

6. Sorting Notes

Each note includes a created timestamp so notes can be sorted.

The app allows sorting by:

- Entry order (creation time)
- Alphabetical title

Each supports:

- Ascending

- Descending (but Ascending is the default)

Sorting only affects **how notes are displayed**, not how they are stored.

This teaches the difference between:

- **Data order**
- **View order**

---

7. Delete and Undo

When a note is deleted:

1. The user must confirm
2. The note is temporarily stored in memory
3. An **Undo Delete** button becomes active

When Undo is clicked:

- The most recently deleted note is restored
- This mirrors behavior in professional applications

This demonstrates:

- State management
- User safety
- Non-destructive design

---

8. Accessibility and Responsible Design

The app follows basic **ADA accessibility principles**:

- <label for=""> is used for form inputs
- Keyboard navigation works
- Focus indicators are visible
- Text contrast is readable
- Screen readers receive status updates using aria-live

Students learn that **accessibility is part of engineering**, not an afterthought.

---

9. How AI Was Used to Create This Program

This program was built using an **iterative AI-assisted process**.

Step-by-Step Collaboration

1. The human designer defined the goal
2. The AI produced an initial version
3. The human tested it and identified problems
4. The AI modified the code based on feedback
5. The cycle repeated until behavior was correct

Examples of AI-guided improvements:

- Fixing accidental duplicate notes
- Separating Save vs Save As New
- Improving sorting logic
- Simplifying the user interface
- Making default behaviors intuitive

Important Lesson

AI did **not** build the program alone.

Instead:

- The human provided direction, critique, and intent
- The AI handled repetitive logic and structure
- Final correctness depended on human testing and reasoning

---

10. Teaching Students to Use AI Responsibly

Students can use AI to:

- Scaffold programs quickly
- Learn patterns and structure
- Generate readable starter code
- Explore alternatives and improvements

But students must still:

- Understand the code
- Test it
- Debug it
- Decide how it should behave

AI is a collaborator, not a replacement for learning.

---

11. Conclusion

This notes application demonstrates:

- How HTML forms collect input
- How JavaScript manages state
- How Web Storage enables persistence
- How thoughtful UX prevents bugs
- How AI can accelerate learning and development

By studying both the **program** and the **process**, students gain skills not just in coding — but in **modern problem-solving with AI**.
