? Week 01: The Personal Dashboard

"Your Digital Command Center"

Welcome to the first real project. In the first week, we aren't just coding; we are reclaiming your browser. Instead of a cluttered "New Tab" page, you're building a lightning-fast, minimalist link organizer that lives on your machine.

The goal for this milestone is to master **CRUD** (Create, Read, Update, Delete) operations and understand how a frontend interface talks to a local database.

?? Phase 1: The Foundation

Before you paste your prompt into an AI, you need to decide on your **Tech Stack**. A good tech-stack is one that lets you ship the fastest, but here are some recommended paths:

Stack	Why choose it?
Next.js + Tailwind + SQLite	The modern industry standard. Fast, sleek, and everything stays in one folder.
Python (Flask) + Bootstrap + TinyDB	Great if you prefer a lighter, more logic-focused backend approach.
Deno + Typescript + HTMX + AlpineJS	Great if you prefer a lightweight stack with simple components that lets you create a single executable binary with the help of demo.
Astro + HTMX + AlpineJS	The AHA-Stack. Simple, minimal and effective.

> **Action Item:** Decide on your language. Do you want to go the JavaScript/TypeScript route or the Python route?

? The Master Prompt

Once you've picked your stack, use this comprehensive prompt to generate the "v1.0" of your dashboard:

```
Build me a personal link dashboard that I'll use as my browser's new
tab page.
[INSERT CHOSEN STACK HERE: e.g., Using Next.js and SQLite]

The app organizes links into categories. Each category has a name and
contains multiple links.
Each link has a name and a URL.

Features I need:
- Display links grouped by category in a clean grid/card layout.
- Add a new link (with name, URL, and category selection).
- Edit and Delete existing links.
- Create and delete entire categories.
- Store everything in a local database (setup instructions included).
- Run on localhost.

Design: Make it clean, dark-mode friendly, and minimal. It must load
instantly.
```

?? Phase 2: Iteration Prompts

Use the next prompts to enhacne your dashboard with more features. Often is less more and small iterations make it easier to get better results.

When you enable git you can also always go back and undo changes.

Also the Planing-Mode in many agents can help to layout a plan before doing any major tasks.

1. Real-Time Fuzzy Search & Filtering

```
Implement a global search bar at the top of the dashboard.
As the user types, it should filter the displayed
categories and links in real-time. Use fuzzy-matching
```

```
logic so searching for 'git' matches 'GitHub' or
'GitLab'. If a category has no matching links, hide the
entire category heading from the view to keep the UI clean.
```

2. Dynamic Favicon & Metadata Fetching

```
Enhance the link display by adding a 16x16px favicon next
to each link name. Generate the icon URL dynamically
using: https://www.google.com/s2/favicons?domain=[URL]&sz=32
Add a fallback 'Earth' icon using your icon library if the
favicon fails to load, ensuring the layout remains
consistent and aligned.
```

3. Persistent Dark Mode & System Preference

```
Add a theme toggle component (Sun/Moon icon). The system
should check for the user's OS preference using
'prefers-color-scheme' on first visit but allow manual
override. Store the chosen theme in localStorage. Apply
a 'dark' class to the root HTML element and ensure all
CSS transitions for colors are smooth (300ms duration).
```

4. Drag-and-Drop Reordering (Persistent)

```
Integrate a drag-and-drop library (like dnd-kit) to allow
reordering of links within a category. When a link is
dropped, send a PATCH request to the backend to update a
'sort_order' integer field in the database. Ensure the UI
updates optimistically so there is no visual lag while
the database saves the new order.
```

5. Data Portability: JSON Backup & Restore

```
Create a 'System' modal that allows data management.
Include an 'Export' button that generates and downloads
a 'dashboard_backup.json' file containing all data.
Also, include a file upload input for 'Import' that
parses the JSON file, validates the schema, and performs
a bulk-insert into the database to restore the setup.
```

6. Smart URL Validation & Auto-Naming

```
Improve the 'Add New Link' form. When a user pastes a URL,
use a regex to validate it. If valid, use a client-side
fetch or server-side route to attempt to scrape the
<title> tag of that website. Automatically populate the
'Link Name' field with this title, allowing the user to
```

```
edit it before saving.
```

7. Keyboard Navigation & "Quick Actions"

```
Implement global 'Hotkeys' for power users. Pressing '/'
should instantly focus the search bar; pressing 'n'
should open the 'Add New Link' modal; and pressing 'Esc'
should close any open modals. Add a small footer or
tooltip that visually reminds the user of these
shortcuts to improve discoverability.
```

? Implementation Tip

When using these, I recommend pasting the **relevant file code** (e.g., your `page.tsx` or `api/links.js`) along with the prompt. This prevents the AI from hallucinating variable names that don't exist in your project.

Revision #8

Created 2026-03-08 17:54:07 UTC by Carsten

Updated 2026-03-13 11:27:18 UTC by Carsten