?? Archiv

Historische Projekte und nicht mehr aktive Arbeitsbereiche. Diese Seiten werden zu Referenzzwecken aufbewahrt, sind aber nicht mehr Teil der aktiven Wissensbasis.

- Project Overview
- Comparison between OneNote and Bookstack, Confluence, Docmost and Nextcloud
- ? Week 01: The Personal Dashboard
- ? Week 02: The Centralized Event Hub

Project Overview

?? Curriculum Roadmap

The bootcamp is structured into 10 weekly milestones, focusing on full-stack development and AI integration.

- **Week 01: Personal Dashboard**
  - **Project:** Personal Dashboard
  - **Goal:** Build a link organizer with a database for a custom browser "new tab" page.
- **Week 02: Build an App with Live Data**
  - **Project:** Events Dashboard
  - **Goal:** Live data visualization using APIs and charts.
- **Week 03: Build an App with Users**
  - **Project:** Shared Expense Tracker
  - **Goal:** Implementing User Authentication (Sign up/Log in) and data ownership.
- **Week 04: Build a Real-Time App**
  - **Project:** Live Chat Room
  - **Goal:** Create interfaces that update live without refreshing.
- **Week 05: Build an AI-Powered App**
  - **Project:** Ingredient Combiner
  - **Goal:** Integrate AI capabilities into your own application.
- **Week 06: Build a CLI Tool**
  - **Project:** Site Inspector CLI
  - **Goal:** Build a terminal tool that inspects any website.
- **Week 07: Build a Paid Product**
  - **Project:** Premium Version of a Previous App
  - **Goal:** Turn your app into a business with real payments.

- **Week 08: Build an Online Store**
  - **Project:** Small-Batch Product Shop
  - **Goal:** Create a complete e-commerce experience with cart and checkout.
- **Week 09: Build an App with AI Agents**
  - **Project:** Link Sharing Community with AI Bots
  - **Goal:** Create AI bots that interact with your app autonomously.
- **Week 10: Build a Complete Product**
  - **Project:** Newsletter Tool
  - **Goal:** Combine everything into one polished, ship-ready application.

? Resources & Community

- **Platform:** Bootcamp Dashboard
- **Community:** Discord (Private Invite Required)
- **Support:** Access to a peer network for troubleshooting and building in public.

? Technical Constraints Note

> **Internal Note:** The student has identified a mismatch between the course content (Cloud/API heavy) and their professional environment (Air-gapped/Offline systems). This Wiki entry serves as a record of the curriculum prior to the refund request on **March 8, 2026**.

Comparison between OneNote and Bookstack, Confluence, Docmost and Nextcloud

I asked:

```
I need a good comparison between Dokuwiki, Confluence, Docmost,
Bookstack and Nextcloud (and a nextcloud app maybe)
We do have Dokuwiki at the moment in our organisation. The new team i
am joining, uses a One Note notebook with hundreds of "books", and then
multiple chapter or cards, and hundreds of pages. some of them really
long.
They want to move away from OneNote. I can think of that they might be
wanting collaborative working, which Docmost and Nextcloud Notes
support.
The biggest issue I see is, that a lot of people do some fancy
formatting which would need to stop when they switch. But effectively
it comes down to: text, lists and images (mostly screenshots) and maybe
a table
Also, the app should have LDAP available (list if paid or freely
included) as this is a key requirement.
Do a throughlouy research on that topic and present me a well
documented thing.
If you ask me, my personal favorite as a long long time dokuwiki user,
would be bookstack, as i myself switch from dokuwiki to bookstack last
year. But this is just a side-note.
Do you need anymore information?
Output should be easily printable or exportable so i can present to my
new team lead at the end of the week.

1. Hosting has to be ON-PREM. We have VMWare and K8s available.
2. Attachments would be nice (i do know that bookstack supports this,
as you can add attachments to pages)
3. mobile is not a requirement. we do not have mobile clients
4. no. it is only used for documentation. if it has a export function
that would be nice.
```

This was the answer:

Understood. I will conduct a thorough comparison of Dokuwiki, Confluence, Docmost, Bookstack, and Nextcloud (including relevant apps), specifically for on-premise deployment within a VMware/Kubernetes environment.

The report will evaluate each tool's suitability for replacing OneNote in a documentation-heavy team, focusing on:

- Ease of use for long-form documentation (text, images, tables, lists)
- Collaborative editing features
- LDAP integration (noting whether it’s free or paid)
- Attachment support
- Export capabilities
- Any significant formatting limitations or user adjustment challenges

I’ll compile the findings into a clear, structured, and printable report for presentation to your team lead. I’ll notify you once the research is complete.

Result:

As the result is really long, I attached it as a pdf to this page. Please open that to see, what ChatGPT has written.

For easy reading, as it is quite a long file, I have just added the Comparisson Results here:

Comparison Summary

To wrap up, here is a high-level summary of each tool’s key strengths and weaknesses in this context:

DokuWiki:

Strengths:

- Very lightweight, easy on-prem deployment, no database needed, numerous plugins, and free LDAP/ACL support. It’s battle-tested and **reliable** for documentation.

Weaknesses:

- UI is dated and not as intuitive for non-technical users (markup editing by default).
- No real-time editing or modern collaboration features.
- Formatting flexibility is limited compared to WYSIWYG editors.
- Migration from OneNote would be manual and formatting might need cleanup.

Good for teams that value simplicity and control over flashy interface.

Confluence (Data Center):

Strengths:

- Rich feature set – excellent editor, powerful macros, best-in-class collaborative editing, attachments handling, and strong enterprise integration (LDAP, etc.).
- It's very **user-friendly** and familiar in feel to Office tools, which helps OneNote users transition.
- Also has the largest ecosystem (plugins for anything from diagrams to workflows).

Weaknesses:

Cost – requires a paid license that can be expensive for on-prem.

- Also resource-intensive to host. Some complexity in administration (upgrades, DB maintenance).
- Another soft consideration: Atlassian's push to cloud means long-term on-prem support is guaranteed only through 2029, but that's still a while.
- Migration from OneNote still not automatic, but Confluence's Word import can ease part of it.

For an organization willing to invest, Confluence provides a robust OneNote replacement with added benefits of structure and integration.

Docmost:

Strengths:

- Modern and feature-rich (almost a drop-in alternative to Confluence/Notion in open source form).

- It offers **real-time collaboration**, a slick UI, built-in diagramming, and Markdown support.
- It's also specifically designed for knowledge bases, with Spaces, page history, comments, etc..
- On-prem deployment via Docker is relatively straightforward.

Weaknesses:

- It's a newer project – still **early stage**, which might mean occasional bugs or missing minor features.
- The biggest consideration is that some enterprise features (especially **LDAP auth**) require a paid license.
- If AD integration is a must and the budget is zero, that's a problem.
- Also, being new, its community and documentation depth is smaller than others.
- Migration from OneNote would be similar to Confluence (no direct import, but can leverage the Markdown/HTML import).

If the team wants a Notion-like experience on-prem and can handle the enterprise feature cost (or doesn't mind local user accounts), Docmost is an attractive option.

BookStack:

Strengths:

- **Ease of use** – very intuitive for users of all skill levels.
- The book/chapter/page structure is great for a documentation-heavy team to organize content logically.
- The WYSIWYG editor and Markdown option cover both bases, and built-in diagram integration is a plus. It's open-source and free, including all features (LDAP, etc.).
- Tags can enhance pages and search significantly.
- On-prem is simple (LAMP stack or Docker). It's relatively lightweight yet capable.
- Export options are excellent for a FOSS tool. A page, a chapter or a whole book can be exported as a PDF, Markdown, HTML or ZIP including images and TOC with clickable links.

Weaknesses:

- Lacks real-time concurrent editing (one editor at a time), which could be a downside if the team frequently co-edits notes.
- Also, the rigid “Shelves & Books” structure might feel constrained if the team prefers a more ad-hoc organization (though one can also not use Shelves if not needed).
- It doesn’t have an official mobile app, though that’s not required.

Overall, BookStack’s **simplicity and user-friendliness** are its selling points, making it likely the least friction for OneNote users aside from the missing freeform canvas aspect.

Nextcloud (Collabora/Tasks/Kanban/Notes):

Strengths:

- Fully on-prem and **integrated platform** – if the team could benefit from other Nextcloud features (file sharing, etc.), this is a big plus.
- Collabora/Notes provides real-time co-editing in a Markdown environment, satisfying collaborative note-taking.
- The 'rich-text' editing is similar to Docmost or Notion which is nicely implemented.
- It’s open-source and includes AD integration for free.
- Also, since notes are stored as files, it’s easy to access or back them up, and even edit via other editors if needed.

Weaknesses:

- The feature set for note-taking is **basic** – no advanced formatting beyond what Markdown offers.
- Users might miss text highlighting, varied fonts, or more visual editing options.
- The UI, while clean, is not as polished or purpose-built for documentation as others (it’s essentially a simple list of pages).
- There’s also no built-in robust export, and navigation is limited to filtering and search, lacking cross-linking ease (though you can link pages manually).
- Another potential weakness is if you’re not using Nextcloud for anything else, you’d be running a comparatively heavy system just for notes – a lot of admin overhead if you only need a wiki.
- Migration from OneNote would likely be manual as well, copying into markdown.

Nextcloud suits an environment where you want a *lightweight wiki* tightly integrated with files and possibly where users already use Nextcloud.

Migration Concerns Recap:

Regardless of tool, expect to invest time in restructuring and copying content from OneNote. OneNote's freeform notes must be linearized, and some formatting (like handwritten sections or arbitrary positioning) won't carry over. Encourage users to embrace the new structure (use headings, use multiple pages instead of one huge canvas, etc.).

There might be an adjustment period where users try to do something "the OneNote way" and it doesn't work – e.g., dragging an image next to text and it doesn't stay side by side. Training and documentation on "how to do X in the new tool" will alleviate this.

On the flip side, they will soon find many advantages: for instance, no more wondering who has the latest version of a note, better search (especially in Confluence, Docmost or Bookstack, where search is quite powerful and maybe even OCRs or indexes attachments in enterprise versions), and the ability for multiple people to contribute easily rather than a single user's OneNote notebook.

? 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.

? Week 02: The Centralized Event Hub

Welcome to **Week 02**! You've already knocked out your personal dashboard; now we're stepping into the world of **Dynamic Orchestration**. This week, you aren't just building a display—you are building the "Nervous System" for every application you will ever write.

? The Mission: Observation & Architecture

The goal is to build a **Centralized Event Dashboard**. This application acts as a private "Log Sink" or "Command Center." It provides a secure API that listens for "Events" from your other apps—whether it's a successful user signup from a web app, a cron job failure in a Python script, or a simple `cURL` message from your terminal.

Build a two-part system:

1. **The Receiver (Remote API):** A cloud-hosted endpoint that is "always on," waiting to catch events from your other apps, scripts, or servers.
2. **The Viewer (Local Dashboard):** A high-performance real-time feed where you can search, filter, and visualize the data flowing through your API.

?? The Architecture

- **Project-Based Isolation:** Manage multiple apps from one hub.
- **API Key Authentication:** Secure your endpoints so only your authorized apps can post data.
- **Persistent Cloud Storage:** Use a cloud database so your data is safe and accessible even when your local machine is off.

? Step 0: Choose Your Stack

Before you run your first prompt, decide on your **Tech Stack**. You will need a database (like Supabase or PostgreSQL) to store your projects and historical event data.

Component	Option A: Modern Serverless (Recommended)	Option B: Robust Python
Backend API	Next.js API Routes (Vercel) or Hono (Cloudflare)	FastAPI (Render or Railway)
Database	**Supabase** (PostgreSQL + Realtime)	**MongoDB Atlas** or Supabase
Frontend	Next.js + Tailwind + Shadcn UI	React (Vite) + Tailwind
Live Updates	Native Supabase Realtime	Pusher or Socket.io

> **Student Note:** Are you a **Next.js + Tailwind** fan? Or do you prefer **Python (FastAPI) + React**? Specify this in your initial prompt so the AI builds the API routes correctly.

? The Starter Prompt

Copy and paste this into your AI chat to generate the foundation.

```
Build me an events dashboard. Other applications send events to it
through an API, and I see them in a real-time feed.

The app has two parts:
1. A REST API that accepts events via POST request (with API key
authentication). This needs to run on a remote server so it's always
available, even when my computer is off.
2. A dashboard that displays events in a feed, with search, filtering,
```

```
and charts. This can run locally.

Each event has: a channel (category like "orders", "signups",
"deploys"), a title, an optional description, an optional emoji icon,
and optional tags.

Features I need:
- POST /api/events endpoint that accepts JSON and stores events in the
database
- API key authentication (generate a key when creating a project)
- A feed page showing events in reverse chronological order
- Filter events by channel
- Search events by title, description, or tags
- At least one chart showing event activity over time
- The dashboard should update in real-time when new events arrive
- Use a cloud database that's always available (Supabase, Convex, or
similar)

Make it clean and functional. I want to actually use this to monitor my
own projects.

Before making any decisions on the stack. Make a stack proposal and ask
me which I want to use.
```

Once you have the initial dashboard frontend, api and database running to your liking, you can continue with the next prompts, to make it better or add additional features to it.

? Evolution: 7 Prompts to Pro Power

Once your API can catch a message, use these iterative prompts to turn a "basic table" into a professional monitoring tool.

The Roadmap:

- **The Real-Time Subscription:** Implement a real-time listener (e.g., Supabase Realtime) to push new events to the feed instantly with a highlight animation.
- **Smart Emoji & Auto-Parsing:** Add logic to automatically assign emojis based on channel names (e.g., ? for orders, ? for deploys) if one isn't provided.
- **Multi-Project Management:** Build a settings page to manage multiple projects, each with its own name and unique API key validation.

- **Advanced Time-Series Analytics:** Integrate Recharts to visualize events per hour and top channels using bar and pie charts across various time ranges.
- **Desktop & Critical Alerts:** Add native browser notifications triggered by specific tags like #error or #urgent, even when the dashboard is in the background.
- **The "Deep Dive" Inspector:** Create a clickable side-drawer for every event to display the raw JSON payload and include a "Copy as cURL" button for debugging.
- **Key Rotation & Security:** Implement a security feature to regenerate API keys, instantly voiding old credentials to protect against leaks.

?? The Detailed Prompt List

1. The Real-Time Subscription

> "Since our database is in the cloud, implement a **Real-time Listener** (e.g., Supabase Realtime). Ensure that when the remote API inserts a new event, the local dashboard pushes it to the top of the feed automatically with a subtle 'new item' highlight animation."

2. Smart Emoji & Channel Parsing

> "Enhance the API logic: if an incoming event doesn't specify an emoji icon, automatically assign one based on the `channel` name (e.g., 'orders' gets ?, 'deploys' gets ?, 'errors' gets ?). Display these icons prominently next to the event title in the feed."

3. Multi-Project API Key Management

> "Build a 'Project Settings' page in the dashboard. Allow me to create multiple projects, each with its own name and unique generated API key. The API should now validate the key against the database and tag the incoming event to the correct project automatically."

4. Advanced Time-Series Analytics

> "Add a 'Metrics' tab. Use **Recharts** to create a bar chart showing 'Events per Hour' and a pie chart showing 'Top Channels by Volume.' Allow me to toggle the time range between the last 24 hours, 7 days, or 30 days."

5. Desktop & Push Notifications

> "Add a toggle in the dashboard for 'Critical Alerts.' If an event is received with a specific tag (like #error or #urgent) or a 'High' priority status, trigger a browser-native desktop notification so I see the alert even if the dashboard tab is hidden."

6. The "Deep Dive" JSON Inspector

> "Make each event card clickable. When clicked, open a side-drawer (Slide-over) that shows the full raw JSON payload received by the API formatted for readability. Include a 'Copy as cURL' button so I can easily replicate the exact request for debugging."

7. API Key Security & Rotation

> "Implement 'Key Rotation' logic. In the Project Settings, add a button to 'Regenerate API Key.' This should instantly void the old key in the database and provide a new 32-character secret to the user, ensuring security if a key is ever accidentally leaked."