

Crafting AI-Driven Web Hierarchy: D3 Force Graphs and the Rule of 7

Setting the Stage: Context for the Curious Book Reader

This piece delves into the practical challenges and innovative solutions behind building an AI-driven, highly navigable website structure. It chronicles the journey of scaling content processing with strategic API key rotation, the technical considerations of moving to a static site, and the exciting implementation of d3 force graphs to visualize a 'Rule of 7' hierarchy. Discover the underlying architecture for transforming flat content archives into a dynamic, user-friendly experience important to know in the Age of AI.

Technical Journal Entry Begins

Okay, so on the current free Google Gemini tier you can create 10 projects per user account and because quotas are against projects and not "global" user accounts, whatever your requests-per-day (RPD) quota limit is, you can multiply that times 10. This is still a fairly generous free tier although you have to work for it a little bit. It's all about friction and convenience, and at some point when you're willing to *pay for the state of the art* because even with all this, it's so that I can get a mere 20 requests-per-day against the gemini-2.5-flash-lite which is far from *state of the art*. But it's just what I need for this task, so we're good. The greater learning from this I think is that with Python it's pretty easy to create "*outer loop*" frameworks to cycle through multiple API-keys so if you're determined to get a lot for free, you can. It just takes a lot of Web UI work to set up those 10 projects per Google login and then the 1 API key per project.

Scaling AI Processing: The Multi-Key Strategy

The meaning of all this is that I'll have all the contextual pre-processing of my articles done today, and that's significant because I also got the stripped-down Jekyll static site generator Minima template ready, though it is barebones in the truest sense. There's *nothing* there and a straight porting of the MikeLev.in site over to that is a potentially deep rabbit hole that I don't know if I'm ready for ahead of the Simon Willison pip install *Ilm refactor* that I've been planning for this weekend. There's a chance I could break it up into a Saturday vs. Sunday thing with today, because it's still actually relatively early at only 2:30 PM. I can do all the little chisel-strike projects I've been putting off like this that come before home-hosting. It's not really worth home-hosting until the *Rule of 7* hierarchy project is ready.

Visualizing Hierarchy: D3 Force Graphs for MikeLev.in

Speaking of which, there's the fact that I have to roll-in a new Widget-type into Pipulate which is the viewing of these hierarchies. Normally I've been using the Cosmograph.app site because it can handle such fabulously large link graphs as I encounter in my day-to-day work. But with this idealized "from scratch" website playground situation, it's a very small link-graph so this opens up other visualization library possibilities, most namely the [d3 force graph](#) that I'm itching to use for this. Okay, let me just work through a little bit of the thought-work for this. Gemini's probably going to know all about the d3 visualizers. Cosmograph also has a more liberally licensable version that allows for a lot more customization [cosmos.gl](#). But I think the AIs are going to be all trained up on d3 and not so much on cosmos.gl, so I know my starting point.

I think it's a new homepage for MikeLev.in with a d3 force graph visualization.

It's going to be the dissection of the website hierarchy issues, really deconstructing and reconstructing all those issues how HTML was intended. Wow, alright. The contextualizing project is all done.

Okay, I have to grab as much context as I can from the prior discussion to carry over into this fresh one, because I'm tired of editing the copied markdown. So with personalization turned off this time starting a fresh discussion should keep that pattern poison from flowing over.

MikeLev.in: Recap with as much detail as you can across as much as you can from this discussion to carry over to another discussion for enough context to smoothly continue this discussion.

The Contextualizer Project: Architecting Content Shards

*Previous Gemini discussion**: Here is a high-fidelity context injection block designed to be pasted into our next session. It captures the technical state, the philosophical goals, and the specific "battle damage" and fixes applied to the codebase.

CONTEXT INJECTION: The Contextualizer & The Rule of 7

1. The Grand Strategy

Goal: Transform a flat, unstructured archive of ~700 blog posts (MikeLev.in) into a structured, AI-navigable tree based on "**Six Click-Ease**" and the "**Rule of 7**" (limit navigation options to 5-9 items per node).

Mechanism: We are generating "Holographic Shards"—small JSON context files—for every article. These files contain metadata, summaries, sub-topics, and keywords extracted by Gemini to later build the edges of the navigation graph.

2. Current Codebase State:

scripts/articles/contextualizer.py

The script has evolved from a simple linear processor to a **multi-key, fault-tolerant batch processor**.

Key Features Implemented:

- **"Hydra Head" Key Rotation:** The script accepts a comma-separated list of keys (e.g., --keys c1,c2,c3). It processes a batch (defined by --limit) using one key. If the quota is hit, it switches to the next key automatically.
- **Quota Loophole:** We leveraged the discovery that Gemini API quotas are *per Project*, not per User. The strategy relies on rotating through ~15-16 distinct Google Cloud Projects to bypass daily limits.
- **"Regex Medic" & Skip Logic:** We encountered json.JSONDecodeError due to LLMs outputting bad escape sequences (e.g., \e).
- *Fix:* Added clean_json_string() to attempt repairs.
- *Fix:* Added strict flow control. If a file fails parsing, it is **skipped** (logged as ) rather than crashing the script.
- **The Compatibility Downgrade:** We attempted to use response_mime_type="application/json". It failed due to an outdated google-generativeai library in the local NixOS environment.
- *Resolution:* We removed the config parameter and reverted to "Prompt Engineering" (Honor System) for JSON generation, relying on the Python parser to catch errors.

3. Current Execution Status

- **Remaining Work:** Approximately 300 articles left to process.
- **Scaling Plan:** The user is creating ~15-16 total Google Cloud Projects/API keys.
- **Math:** 20 articles per key × 15 keys = 300 articles (One "Super-Run" to finish the backlog).
- **Last Successful Test:** Processed a batch with key rotation. Quota stops are handled gracefully; parsing errors are skipped gracefully.

Current Status and the "Super-Run" Scaling Plan

4. The Code Snapshot

The script currently looks like this (critical logic for the next session):

```
# Key Logic in generate_context_json:
# Returns status_code: 0=Success, 1=Quota (Stop Key), 2=Error (Skip
File)
# Uses strict try/except blocks around json.loads()
# Does NOT use generation_config={"response_mime_type":
"application/json"} (Keep this off!)

# Key Logic in main/process_batch:
# Iterates through `keys_queue`.
# Slices `to_process` list based on `--limit`.
```

```
# If `process_batch` returns early due to quota, the outer loop
switches keys.
```

5. Next Immediate Steps

1. **Execute the "Super-Run"** with the full array of 16 API keys to finish the remaining 300 articles.
2. **Phase 2 Begin:** Once all .json context files exist, begin designing the **Navigational Graph**.
3. **Task:** Algorithmically group these JSON shards into the "Rule of 7" hierarchy (Parent -> Child nodes) to replace pagination with drill-down navigation.

6. Constraints & Environment

- **OS:** NixOS (Immutable, specific library versions).
- **Library:** Older google-generativeai (does not support response_mime_type).
- **Philosophy:** "Sovereign Technomancer." We prefer local control, flat files, and creative workarounds (like key sharding) over paying for enterprise tiers or accepting default limitations.

MikeLev.in: Alright Gemini. That should catch you up from the previous article. Provide whatever commentary and then also show me your understanding of the data file I would have to provide for the d3 force graph visualization.

Gemini: This is an excellent "chisel-strike" project. It's self-contained, high-visual-impact, and perfect for testing your "Rule of 7" logic before you commit to a heavy backend architecture. You are correct that **d3-force** is the standard here. It has a steeper learning curve than cosmos.gl, but it gives you absolute control over the physics, which you need to enforce that "cluster-like" behavior where nodes feel magnetic to their parents. Here is the exact data structure understanding you need.

The Target File: graph.json

To power a standard **d3 force-directed graph**, you need a single JSON object containing two arrays: nodes and links.

D3 is physics-agnostic; it just wants to know **what exists** (nodes) and **what is connected** (links). The "Rule of 7" hierarchy is defined entirely by which links you generate.

```
{
  "nodes": [
    {
      "id": "root",
      "label": "MikeLev.in",
      "type": "hub",
      "val": 20
    },
    {
      "id": "python-category",
      "label": "Python Automation",
```

```

        "type": "category",
        "val": 10
    },
    {
        "id": "article-slug-1",
        "label": "Pipulate: SEO Automation",
        "type": "article",
        "val": 5,
        "url": "/pipulate-seo-automation"
    },
    {
        "id": "article-slug-2",
        "label": "The Python Outer Loop",
        "type": "article",
        "val": 5,
        "url": "/python-outer-loop"
    }
],
"links": [
    {
        "source": "root",
        "target": "python-category"
    },
    {
        "source": "python-category",
        "target": "article-slug-1"
    },
    {
        "source": "python-category",
        "target": "article-slug-2"
    }
]
}

```

The "Bridge" Logic (Where the Magic Happens)

Your current status is that you have **700 flat JSON shards**. You cannot simply feed those shards to D3. You need an intermediate Python script (a "Graph Builder") to ingest those shards and spit out the graph.json above.

Here is how your data maps to the D3 requirements:

1. The Nodes Array (Easy)

This is a direct mapping of your Shards.

- **id**: The unique slug of the article (e.g., 2024-12-13-python-loops).
- **label**: The Title (from your Frontmatter/JSON).

- ****group / type****: Useful for coloring. Is it a "Category Hub" or a leaf "Article"?
- **val**: The visual size of the bubble. Hubs should be bigger (20) than articles (5).

2. The Links Array (The "Rule of 7" Logic)

This is where you win or lose the game. A standard force graph is just a "hairball" mess. To get the "Rule of 7" structure, your Python script must decide **who the parent is**.

- **source**: The ID of the Parent (e.g., "python-category").
- **target**: The ID of the Child (e.g., "article-slug-1").

The Critical Decision:

Since your current shards are likely flat (just articles), you probably don't have explicit "Category Node" shards yet. You will likely need to **synthesize** these Category Nodes in your graph.json generation script to serve as the gravity centers for your articles.

Visualizing the "Rule of 7" in D3

To make the graph look like a hierarchy and not a net, you will tweak the D3 simulation forces later (I can help with this code):

D3 Physics for the "Rule of 7" Visualization

1. **d3.forceManyBody**: Negative charge (repulsion) so nodes don't overlap.
2. **d3.forceLink**: Pulls children to parents.
3. **d3.forceRadial (Optional)**: Can force specific hierarchies into concentric circles, which looks very "Rule of 7".

Next Step

I can write the **Python "Graph Builder" script** right now.

It would scan your `_site/` or `_posts/` folder for the .json shards, identifying common keywords/tags to create ad-hoc "Hub Nodes" (to satisfy the Rule of 7), and output the final graph.json for D3.

Shall I create that Python script to turn your flat shards into a hierarchical graph.json?

The "Two Rule of 7s" Hybrid Navigation Architecture

MikeLev.in: Okay, right. So first of all the bad markdown is still happening. For your edification and hopefully to run it up the flagpole that something is terribly broken, when you generate a headline response, the # of the headline "touches" the text that begins after it (with no space) and that horribly breaks markdown. What's worse is that the paragraph following that is concatenated directly to the end of where the headline ends so that to get it to format correctly, the human has to find where the headline stops and the paragraph begins and insert 2 line-breaks to get it correct. It makes publishing your output cleanly very, very tedious. I thought I caused this with a bad prompt but after turning off Gemini personalization (seeing prior discussions) and still seeing this problem, I think it's something Google broke in the chat framework. This is just an FYI. I just deal with it until it's fixed, but if you have flagging system in your bag o'tool-calls in your bag o'tool-calls, this needs flagging.

Anyhoo, onto the main event. You're right about where this is won or lost. And for starters I'm just going to show you (already did because of how `prompt_foo.py` works) how I'm summarizing the articles, keywording them and finding sub-topics. I'll give you a small sample just to prime the pump or warm the cache, as they say. This is not the request to figure out the hierarchy yet, but it should give you a taste of how I sized all the article summaries to fit easily in your 1-million token context-window. I'm only at about 720 articles and I calibrated the system to fill you up at 5,000 articles so we're still in the shallow end of the pool.

This should be just enough to wet your appetite, see how it works, be relieved that I'm not going to shove all that "full article" context into you, and ideate what's possible going from the lightweight JSON for all ~720 of those articles to a *Rule of 7* version of that d3js structure. Every level of drill-down has TWO *Rule of 7* things: 1 for the articles at that actual level and another for the further drill-down into the hierarchy. Does that make sense?

Note: The thought-process is all-important here, so we capture it:

Show thinking

Conceptualizing Hierarchy Structure

I'm currently focused on the practical implementation of the "Rule of 7" for structuring the blog post summaries. My goal is to create an effective, navigable hierarchy suitable for visualization. I'm actively considering how best to group related summaries and how to translate those groupings into the graph's nodes and edges. It's a challenging endeavor, but I am committed to producing a visually appealing, informative d3 graph.

Refining JSON Mapping Logic

I'm now deep-diving into the JSON structure and the "Two Rule of 7s" constraint. I'm exploring how to translate the flat JSON shards into a hierarchical structure for the d3 graph. I'm meticulously reviewing the provided JSON keys and how they relate to potential parent and child nodes. My focus is on creating a balanced hierarchy with meaningful connections for the visual representation. I'm taking extra care to avoid any markdown formatting errors.

Constructing the Hierarchy

I'm now implementing the "Two Rule of 7s" for articles and categories, aiming for a visual representation. The goal is ~7 articles AND ~7 sub-categories per node, creating two distinct lists. The key is to organize the flat JSON shards into this hierarchical structure. I'm focusing on node assignments and hierarchical depth, ensuring the graph is not too deep or flat. The JSON structure has several helpful keys, but the work is not yet finished.

Defining Hybrid Node Structure

I'm presently zeroing in on the "Two Rule of 7s" and translating this constraint into a hybrid node setup for the D3 graph. I've realized that each node will encompass both direct article links and sub-categories, providing a more intuitive navigation experience. This design eliminates unnecessary category layers. I'm now considering the best method to output the nodes and links in the required JSON format for D3, ensuring visual clarity and adherence to the "Small World" concept.

Clarifying Hub Node Synthesis

I'm now deeply engaged in the "Hub Node" synthesis for the D3 graph, recognizing its absence in the source JSON. My approach centers on a "Semantic Clustering" technique, relying on the textual similarity within the kw and sub fields since the data doesn't include embeddings. Exact string matching is fragile, so I'm considering using partial string matching and perhaps some light text pre-processing to group related concepts. I'm focusing on the algorithmic strategy to define and populate these hub nodes effectively. The contextualizer script and example JSON files are crucial references.

Optimizing Hub Node Generation

My focus has shifted to the practical application of keywords for generating Hub Nodes. I'm initially considering frequency analysis to extract the top keywords, but a hybrid approach with simple set intersections to group by similar keywords seems like a practical first step. However, leveraging the existing ILM library might enhance the taxonomy creation, though not as a primary approach, to generate once and not as a default. Ultimately I'm aiming for a deterministic, fast method to form Hub Node relationships.

Devising Recursive Clustering

I'm now implementing the "Recursive Frequency Clustering" algorithm as the core of the graph builder. This algorithm takes a decision-tree approach to site navigation, specifically addressing how to handle articles exceeding the "Two Rule of 7s" constraint. I'm focusing on the implementation details for the recursion step to handle these overflow articles. I'm also paying close attention to the user's feedback regarding markdown and am incorporating checks to ensure proper formatting of headers. I'm confident this method will produce a well-structured and navigable d3 graph, given the JSON schema.

/Show thinking

Gemini: This makes perfect sense. You are describing a **Hybrid Navigation Node**.

Instead of the traditional "Folder" (which contains *only* other folders) or "File" (which is a dead end), you are proposing a "**Hub & Spoke**" model for every single node in the hierarchy.

The "Two Rule of 7s" Architecture

Here is how I visualize the data structure you are describing. Every circle in your d3 graph is not just a point; it is a **Station** that offers two distinct departure boards:

1. **The Local Board (Content):** "Here are the 5-7 best *articles* that live exactly at this level of abstraction."
2. **The Express Board (Drill-down):** "Here are the 5-7 *sub-topics* if you want to go deeper."

This solves the "Empty Room" problem of traditional file trees where you have to click five times just to find a document. In your system, there is content *everywhere*.

From Shards to Structure: The "Graph Builder" Strategy

We have ~720 flat JSON shards. We need to synthesize a tree. Since we are avoiding the "Big Vat of Content," we cannot just dump them all into root.

Here is the algorithmic approach to building your graph.json using the metadata you already extracted (kw and sub):

1. The Clustering Pass (Synthesizing Hubs)

We don't have explicit categories yet, so we must invent them based on frequency and gravity.

- **Scan:** Read all 720 shards.
- **Tally:** Count the frequency of every kw (keyword) and sub (sub-topic).
- **Elect Leaders:** The top 7 most frequent terms become the **Level 1 Hub Nodes** (e.g., "Python", "Linux", "AI", "SEO").

2. The Assignment Pass (Gravity)

We iterate through the articles again.

- **Assign:** If an article contains the keyword "Python," it gets pulled into the "Python" Hub's

orbit.

- **The "Rule of 7" Check:**
- If the "Python" Hub has ≤ 7 articles, they stay on the **Local Board**.
- If it has > 7 articles (which it will), we recursively repeat the "Clustering Pass" *only on the articles inside the Python Hub*.
- We identify the top 7 *sub-keywords* within Python (e.g., "FastHTML", "Pandas", "Automation") and create **Level 2 Hub Nodes**.

3. The "Orphan" Handling

There will always be articles that don't fit the big clusters.

- **Strategy:** We create a "Miscellaneous" or "Latest" Hub for each level to catch the stragglers, ensuring they are accessible without polluting the main navigation.

The D3 Visualization: Physics of the "Rule of 7"

When we feed this into D3, the "Two Rule of 7s" translates into specific physics forces:

1. **Cluster Gravity (forceManyBody):** Articles will clump tightly around their parent Hub.
2. **Radial Constraint (forceRadial):** We can arrange the **Express Board** (sub-topics) in a wider outer ring, and the **Local Board** (articles) in a tighter inner ring around the Hub.
3. **Color Coding:** * **Hubs:** Dark/Solid (Visual Anchors). * **Drill-downs:** Lighter shades of the parent color. * **Articles:** Distinct color (perhaps white/grey) to signal "Read Me."

The "Sovereign" Advantage

Because you generated these shards yourself using `prompt_foo.py` and the contextualizer, you own the taxonomy. You aren't relying on Google's "Search Generative Experience" to group your content; **you are defining the map**.

This is the perfect dataset to feed into the **Universal AI Interface** (the `llm` refactor) we discussed. You can eventually ask the AI: *"Look at these 100 articles in the 'Python' cluster and invent the 7 best sub-category names for them,"* instead of relying solely on keyword frequency.

Strategic Graph Building for AI-Powered Taxonomy

Status: The cache is warm. The logic holds. The "Two Rule of 7s" is a viable graph constraints model.

Shall we proceed with the **Universal AI Interface** refactor (`pip install llm`) so that you have the robust tooling to build this graph?

MikeLev.in: No. Here I rest to contemplate. I'll be back to this soon.

Book Analysis

Ai Editorial Take

This entry serves as an interesting blueprint for integrating AI-driven content contextualization

with advanced visualization techniques for web architecture. It powerfully illustrates how strategic problem-solving (like API key rotation) enables ambitious AI projects. The 'Two Rule of 7s' concept for d3 is a particularly insightful contribution, offering a novel way to manage information density and navigation in a content-rich environment. It perfectly sets the stage for future discussions on autonomous content organization and interactive data representation.

Content Potential And Polish

- **Core Strengths:**
- Demonstrates a highly practical and ingenious approach to overcoming API limitations for large-scale data processing.
- Articulates a clear, innovative vision for content organization via the 'Rule of 7' and d3 force graphs, bridging abstract principles with concrete visualization.
- Provides excellent, detailed technical context regarding the contextualizer.py script's evolution and current state, crucial for understanding the data foundation.
- Showcases a 'Sovereign Technomancer' philosophy, valuable for readers seeking self-sufficiency in AI-driven projects.
- **Suggestions For Polish:**
- Expand on the early design considerations for the 'Rule of 7' algorithm, particularly the initial challenges in grouping articles before the 'Clustering Pass' was devised.
- Include a conceptual diagram or sketch (even textual) of the 'Hybrid Navigation Node' to further clarify the 'Two Rule of 7s' architecture for a non-technical audience.
- Briefly explain prompt_foo.py's role earlier in the piece to provide immediate context for how the JSON shards are initially generated.
- Add a forward-looking statement about how the D3 visualization might eventually interact with an AI agent for real-time adjustments or user-specific navigation.

Next Step Prompts

- Given the 'Two Rule of 7s' and the 'Graph Builder' strategy, provide pseudocode for the process_batch function within graph_builder.py that handles the recursive clustering and assignment of articles to hybrid nodes.
- Elaborate on the specific d3.force* configurations (e.g., strength, distance) required to visually represent the 'Two Rule of 7s' architecture, providing a small JavaScript snippet for context.