Dear Diary!

Now that I have a documentation index, I could write a server, that serves that index and searches it. That server will ultimately also deliver my frontend application, because I'm too lazy to do it the microservices way right now.

The most easy and stable web server for Node I know today is **ExpressJS** and that's what I used.

I'm mostly doing my projects in Typescript today, so that was my basic setup.

After the index has been created or read from my pre-rendered index, the **server** starts up, registers its endpoints and listens for requests.

One **very simple controller** would basically just output various areas of the index:

- **the complete index**
- **All available vendors**
- **The available object types (datasources and resources) for a vendor**
- **All datasources by the vendor** and **all resources by the vendor**

The interesting part came with the search endpoint.

I could simply return all items that include a certain query, but I wanted *weighted* results.

See, I'm no search engine expert and other people can do that way better, but I tried.

How could I weigh the search results? I needed to make a **decision workflow** for that and weigh upon the results

- Datasources and resources are much more relevant than Vendors
- Resources are more relevant than datasources (yes, that one is highly opinionated)
- Matches in names are more relevant than in descriptions
- If a match is found in the vendor **and** an object, the result becomes more relevant

Additionally, I blatantly steal the **string ranking algorithm** of the wonderful **Keeweb project**

I guess, that's as good as it gets currently.

Yours

Dennis

*This post is one of five posts from the **tflookup developer diary series***

*Cover Image: **„diary writing“ by Fredrik Rubensson***

*Originally published to **dev.to***