Prologue

I have established some kind of „learning project“ some time ago. A fixed set of features of an application that I reapply to things I want to learn from time to time.

The background of that application is that I’m using e-mail aliases for each service I subscribe to. So I have a different e-mail for Amazon and Github for example.

I’m using an OpenLDAP server for this configured into a Postfix mta to achieve this.

This learning project consists of a frontend for managing these aliases. Thus, it is always called *Aliasmanager*.

Introduction

Like the title suggests, this is the third time around I’m starting on a green field with Aliasmanager.

The first one used **Qooxdoo** for the API-Server and the frontend, but it was kinda monolithic. The second one had an **AngularJS** frontend and a **Flask** API-server, but still both parts somehow were merged together.

For the third version I wanted to go microservices. At *least* develop the API-Server completely separated from the frontend.

What I really like about Flask is that it is a progressive framework. It starts really small and you can add all the things you need and simply plug it in.

Another framework that does the same thing is **Nest**.

The API-Server

So this time, the API-Server was made using Nest. Nest has a really nice way to structure your application into multiple modules and separates services from controllers. At one point I thought about using a MongoDB backend and that was also very easy to incorporate into Nest, but I skipped that idea and got back to using only the LDAP-Server.

Nest has some pretty nice integrations. For example, authentication and authorization is pretty easy as it integrates into **Passport** and I really only needed to do some lines of code to be able to have an LDAP authentication for the login endpoint and a secure JWT-based authentication on the other endpoints.

Also, documenting the API was very ease due to Nest's integration of the OpenAPI specification.

The only thing I found hard was that Nest's documentation is somewhat blank in some parts and I really would've liked some kind of API documentation.

But in the end everything went fine and I also really enjoyed the **Jest** testing integration.

Using **Dockerode** I was able to remotely manage my docker daemon to ramp up an OpenLDAP server (based on **Osixia/OpenLDAP**) for the test suites.

However, that resulted in some challenges later.

But read on.

The Frontend

This time I switched to **Vue** and while Vue itself was a breeze (I know Vue for some time now and really enjoy developing in it), I've actually tried three (!) component frameworks until I got it right.

The first one was **Vuetify** and while I really enjoyed the Material Design Vuetify uses, the documentation and certain small aspects is a mess. Various parts did not work as described in the documentation or worked differently. For example, I was expecting to have an event

triggered when a modal was shown so I could set the focus on one of the input fields. But that was not the case.

The second one I tried was **Bootstrap-Vue** and here, mostly everything worked as expected and the documentation was okay. However, I didn't like the design that much and the a11n support was somewhat problematic.

The last thing I tried and stuck with was **Buefy**, an integration of the Bulma CSS framework for Vue. There still were some smaller things (like the mobile view acts a bit weird around the edges), but so far everything was fine and I liked the design much better.

A11n

Also, as I mentioned before, I spared some thoughts about accessibility (or a11n) as I think *every* frontend developer should do these days. I didn't go as far as I should have, but I used **Andi** to check for possible a11n flaws. For some part, this was a bit egoistic, because I wanted a good keyboard navigation. And when you have that and good labels for your elements, you're half the way to a good a11n. Please correct me, if I'm wrong about that.

Frontend testing

Another important thing was testing the frontend and Vue incorporates **Cypress** for proper end-to-end testing and I absolutely like how easy Cypress makes frontend testing having a background in Selenium.

One thing, I still didn't get right was code coverage reporting combined with Vue and Cypress. I know about **babel-plugin-istanbul** and after babel's cache drove me mad a couple of times I kinda had some code coverage statistics, but they still don't look quite right.

Getting it all together

I'm running the Aliasmanager on my single machine, so I needed a way to bring all parts together and obviously, this is done using Docker and docker-compose for my single machine or even Kubernetes for clusters.

So I made Dockerfiles for each component and just right before finishing it all I saw that little note „Autotests disabled" in Docker hub's autobuild-feature and I checked that out.

And it is actually pretty nifty.

Docker uses **containers for running tests on pushes and pull requests**. You simply add a new docker-compose file called docker-compose.test.yaml and place a „sut" service into that and docker will run it.

Pretty easy, huh?

Well, yes. It is.

Until your test suite relies on Docker for an LDAP sidecar container.

So I needed to turn to a „Docker-in-Docker"-like infrastructure and control that from Dockerode in my test suite.

For example, I needed the ldap container to be reachable from my test container. As docker-compose creates its own docker network, I had to find out which network it had created and connect the ldap container to that same network.

The second thing was a seed file I needed to fill my ldap server with sample data for the tests. It had to be a file as the image does not support doing this using environment variables and this file was on my machine and in the test container but how to get it into the ldap container from *within* the test container?

The solution was to use a separated volume and also bind that volume to the ldap container.

So that whole docker testing part was actually the hardest part of all.

Conclusion

I learned a lot this time. From using Nest, to component frameworks for Vue to good testing using Jest and Cypress and advanced autobuild and -test features of Docker hub.

You can check out all the parts here:

- **Api-Server**
- **Frontend**
- **Docker-compose**

Photo by **Chad Peltola** on **Unsplash**

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