

umass-cs-497s-F22 / **developer-notebook-jackbisceglia** lines Stat is unavailable **Private**

<> Code Issues Pull requests 1 Actions Projects Security Insights

developer-notebook-jackbisceglia / week-06 / README.md in main

<> Edit file

Preview

Developer Notebook Week 06

Here's a comprehensive breakdown/blog/notebook of what my group and I worked on this week when completing our project milestone 1!

Idea Brainstorming

First, we brainstormed ideas. I may have talked about this already in the last dev notebook entry, but this definitely got more in depth this week as we hit crunch time for the project deadline. I created this website to track our ideas, <https://497-idea-sheet.vercel.app/>, and we ended up going with this idea: Anonymous Feedback Links

There were a lot of good ideas posted, and some good input from the team on this website. I mostly just did this because I was bored, but I think it ended being helpful, at least for me to prioritize what ideas we had and if we should brainstorm further. It helped also to prioritize when looking at which ideas would be most conducive to microservice architecture.

Once we got our idea, we whiteboarded in the library quite a bit. I don't have the whiteboard images on hand (I think Joe does, and they're in our milestone submission), but it was a nice preliminary breakdown of what we are creating.

This was really the bulk of the brainstorming phase.

Docker

Now we came to Docker. The first thing I did was look into booting up a basic container. I followed along with the official Docker documentation for Golang. First I made this simple web server (pretty much just this 1 file, save for module stuff)

```
package main

import (
    "fmt"
    "io"
    "net/http"
    "os"
)

func main() {
    httpPort := os.Getenv("HTTP_PORT")
    if httpPort == "" {
        httpPort = "8080"
    }

    fmt.Println("Hello, world!")

    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        io.WriteString(w, "Root path running")
    })

    err := http.ListenAndServe(httpPort, nil)
    if err != nil {
        fmt.Println("Server Closed")
    }
}
```

Once I had this, I created my Docker file (really just completely the same configuration from the Docker Golang documentation):

```
# syntax=docker/dockerfile:1

FROM golang:1.16-alpine

WORKDIR /app

COPY go.mod ./
COPY go.sum ./
RUN go mod download

COPY *.go ./

RUN go build -o /testdocker

EXPOSE 8080
```

```
CMD [ "/testdocker" ]
```

Then I built and ran, and had things working!

Further, during Thursday's class I read through this post: Building Efficient Dockerfiles - Node.js about efficient Docker files as I was trying to understand the reasoning for installing modules before copying over source code (Ashir sent it to me):

Auth Patterns in Microservices

As for planning out our application architecture, as expected, one point of interest/concern for the whole team was authentication. I did quite a bit of research as to how we should handle this. Out of impulsion (and a need to fulfill 12 services), we planned on having an authentication service. Luckily, after reading this article, A different approach to User Sessions in Microservices, it looked like our guess/assumption is valid. My mental model in our assumption is that the Authentication/Authorization service would act as essentially a middleware/proxy between our original request (generally speaking, the client), and any protected paths. Examples for us would be creating rooms, seeing past rooms, etc-- these should be verified as authenticated by our auth service before passing through to the actual service. The aforementioned article seems to indicate that this is the way to do things, assuming we don't go by way of token-based authentication

I also kind of checked out this article as well, but it was more detailed, complex, and long, so I plan on revisiting this in the future: <https://doordash.engineering/2021/05/28/session-management-migration/>

Client Side Authentication Patterns For the sake of authentication in its entirety, which I often find myself having trouble with, I thought it would be wise to look into some common patterns of authentication handling on client side. In the past I've kind of just stuck with a Server-Side approach, where I request HTML from the server, and before returning it, I verify that the user is authenticated. However, in the case of a Single Page Application like we've viewed in class, this idea of Server Side HTML serving is kind of off the table (unless we use Next.js :). Nonetheless, I've found this GitHub repository from Ben Awad: [lireddit](#), to be really helpful in the past, so I revisited it. Looking through his logic, it looked as if his primary approach for the client-side auth pattern was, authenticate our session with the user after the page has loaded its HTML. In hydrating the page, block the loading of any data, until we've ensured that our user is authenticated with a server response. He extracts this into a custom `useIsAuth()` hook where he can reuse on many protected pages. The logic is below

```
import { useMeQuery } from "../generated/graphql";
```

```
import { useRouter } from "next/router";

import { useEffect } from "react";

export const useIsAuth = () => {
  const { data, loading } = useMeQuery();
  const router = useRouter();
  useEffect(() => {
    if (!loading && !data?.me) {
      router.replace("/login?next=" + router.pathname);
    }
  }, [loading, data, router]);
};
```

I think our data flow can be largely similar client side, assuming our auth service works as expected.

Best Redis Use-cases

Lastly, I briefly looked into potential Redis use cases, a message queue among many others, and this article (from 2010!) came up: Redis Tutorial. It seems to be really good and hit on the key points and aspects of Redis and what it intends to accomplish. My main question was if we can simply store a key, not a value, (so pretty much an array), that way many of our services can simply check if a service exists. If it does, it's active. If it doesn't, either it never existed, or it has passed its expiration. The answer, from what I gathered, is that this is indeed possible.

Other Web Stuff

I saw this video linked on LinkedIn or something like that:


<https://www.youtube.com/watch?v=u8mvpZq3RGc&t=12s>

I thought it was pretty interesting. Of course I understand the power of microservices in the real world for scaling teams, and the purpose of them for this course for the sake of education, though I do think it's interesting to also keep in mind other scopes of these things, such as how would the founder of a new software product assess the adoption of a microservice architecture.

Commit changes

Create README.md

Add an optional extended description...

- ☐ -o- Commit directly to the `main` branch.
- ☒  Create a **new branch** for this commit and start a pull request. [Learn more about pull requests.](#)

Commit changes

Cancel