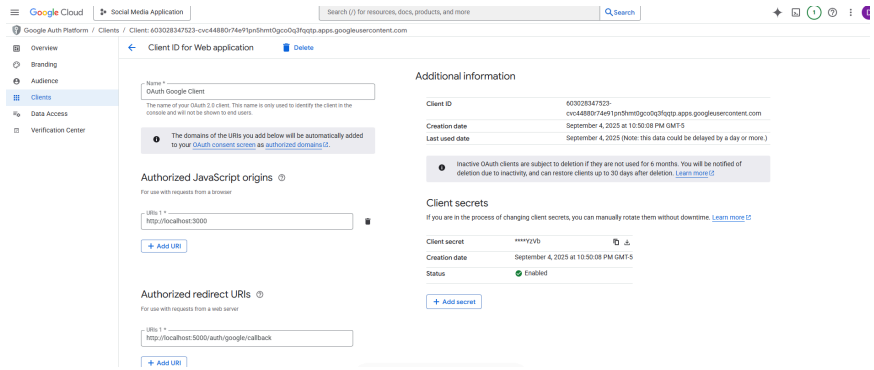
Introduction:
I went with a monolithic architecture for this repo to get it off the ground faster. The frontend is javascript using react framework and the backend is using node express with a mongoose NOSQL database. I have set the app up in a way where it could easily be deployed to vercel for the frontend and heroku for the backend in the future.

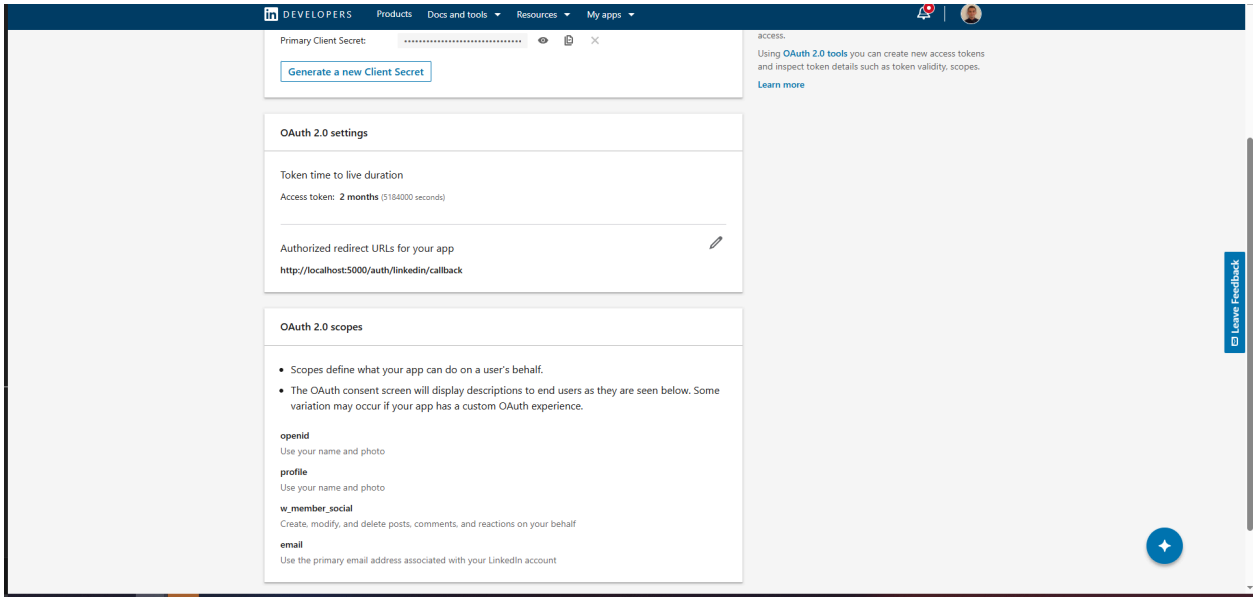
1. Passport: https://www.passportjs.org/
2. https://github.com/Cazman06/Social-Media-Application

Development Process/TroubleShooting:
I started with investigating a library for OAuth 2.0 that I could use and found passport so I then based my tech stack off of that. I started with the frontend setting up the boilerplate and then created the login page and home page utilizing MUI for components and styling. I then moved onto the backend starting with google and set up the google app like so:



After that I wrote the code to register the google strategy and then execute that strategy through passport's use and authenticate methods. My original idea was to do facebook for the second login but struggled with not having to submit it for a review since it was a business application and I wasn't sure how long that would take so I switched to trying linkedin. LinkedIn's documentation was not up to date and I tried using the regular linkedin OAuth 2.0 flow but parts of that seemed to be deprecated so I ended up having to go with passport-openidconnect and

set up the open id part on my LinkedIn app like so:



I made sure to send correct scopes matching this with my app and then was getting an error saying my issuer was not correct. The kicker with that though was it was silently failing when it was trying to execute the verify callback so I had to create a temporary custom logger within my passport.authenticate() call to see that error message and control clicked to get to the associated node module file to make sure my parameters were right. Once I was sure of that, I then used this link: https://www.linkedin.com/oauth/.well-known/openid-configuration to check the open id config and noticed the issuer had /oauth at the end, I didn't find this in the docs so I updated that and was able to have successful logins as shown in the video. After that I went to implementing process.env and made sure to have the id's and secrets in that so it's more secure. Ideally the secrets would be stored in something like AWS secret's manager, not an env.
Backend Env For Copying:

```
GOOGLE_CLIENT_SECRET=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
LINKEDIN_CLIENT_ID=86ulknwt849gxw
LINKEDIN_CLIENT_SECRET=WPL_AP1.sF7InO0kpvMxBEOU.0kkIeQ==
SERVER_URL=
SESSION_SECRET=aVerySecret
MONGODB_URI=mongodb+srv://cazman06212_db_user:0q8sYs0k3i3Dmh6R@cluster0.0d
580v8.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0
```

Finally I went to error handling and made sure I had ideal errors thrown during database connection instantation and for the verify callback from passport.

Architecture Diagram: