Software Design Document

Dynasty Daddy

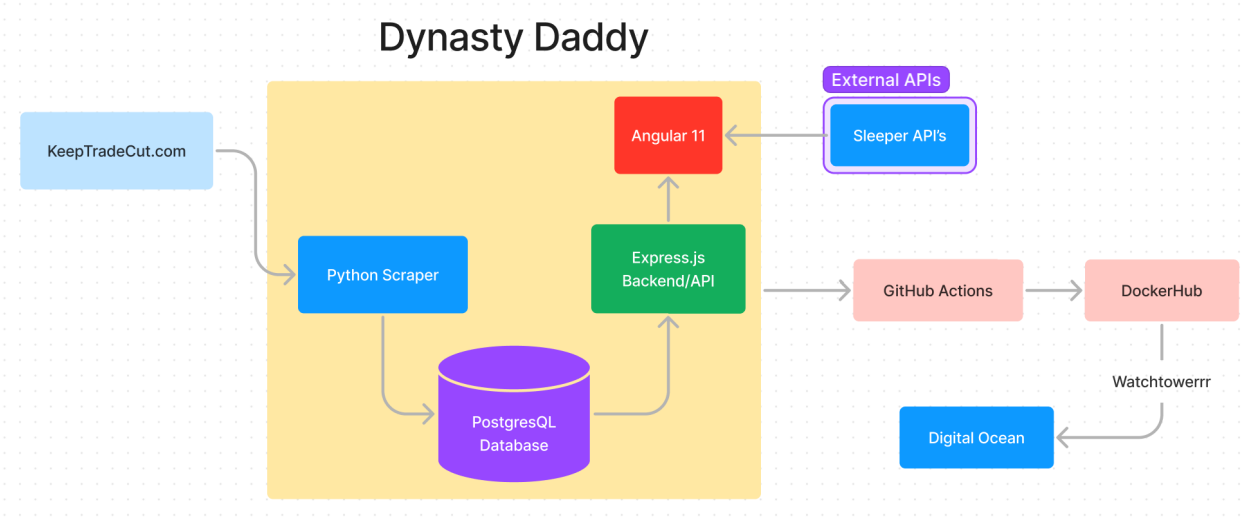
Date: May 19th, 2022

Written by: Jeremy Timperio

Introduction

Dynasty Daddy is a web app that integrates with Sleeper's API and scrapes KeepTradeCut's player evaluations to create metrics on each owner's fantasy league. The goal is to provide users with a frictionless way to see each player’s value, team value, draft breakdowns, and more. When managing my teams I would constantly have to switch between multiple websites to figure out what moves I should make. I wanted to find a way to spend less time doing research and leverage the data in more ways than before. Thus, Dynasty Daddy was born to help me beat my friends in fantasy easier! This document is to provide a written breakdown of most of my important decisions and thought processes as I put together this application.

System Overview



Dynasty Daddy is an Angular, Node, Express, and Postgres application. The intention is to have a site where users will be able to enter a league id or sleeper username to pull league-specific data. In order to do this, I needed to integrate Sleeper's APIs into my application to pull specific fantasy league data. Another important external requirement was to be able to pull up-to-date

player values from KeepTradeCut.com (a crowdsourced player value elo website). Unfortunately, KeepTradeCut does not provide public APIs as of now. In order to work around this, I introduced a python script to scrape this site every day and update the Postgres database with current player values.

Design Considerations

Assumptions & Dependencies

- **Sleeper API:** Sleeper's API suite is an external dependency. While they haven't changed the v1 endpoints for years, a change to the APIs would break the Sleeper integration.
    - Note: It looks like Sleeper has changed to use graphql going forward so the v1 APIs are most likely legacy.
- **KeepTradeCut Classes:** In order to scrape KeepTradeCut, I use their class definitions in the HTML to pull the values. If this information/design would change, that would break my daily scraper of KeepTradeCut data.

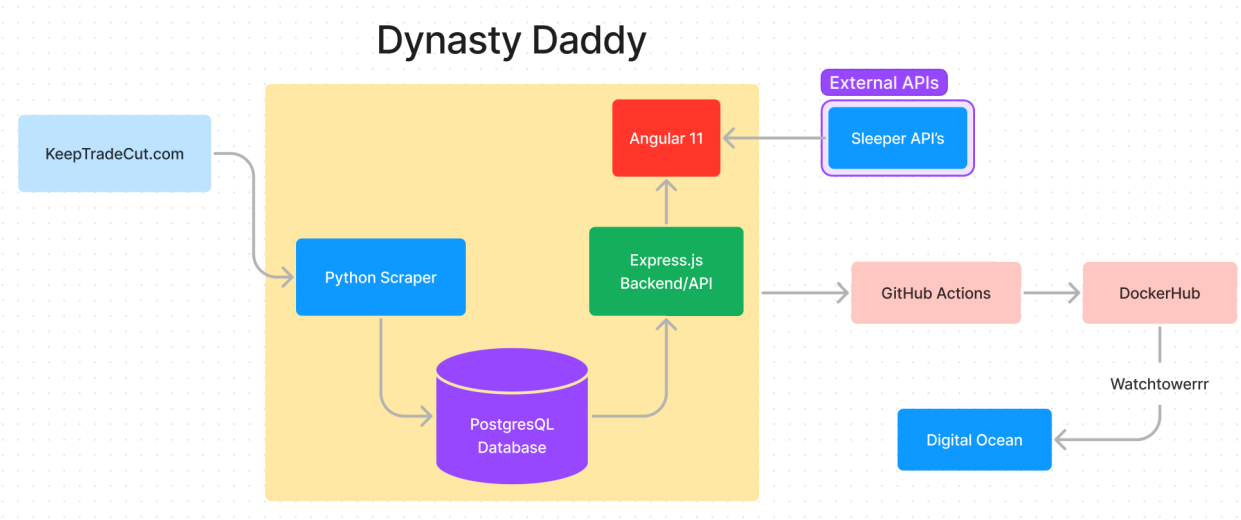
General Constraints

- **Developers:** The largest external constraint is there is only one developer currently, me. Work comes in waves based on how demanding my job, life responsibilities, and motivation for this project is.
- **Money:** Financial backing limits many architectural decisions. As a way to save money, there is no deployed development environment and limits on scalability.

Development Methods

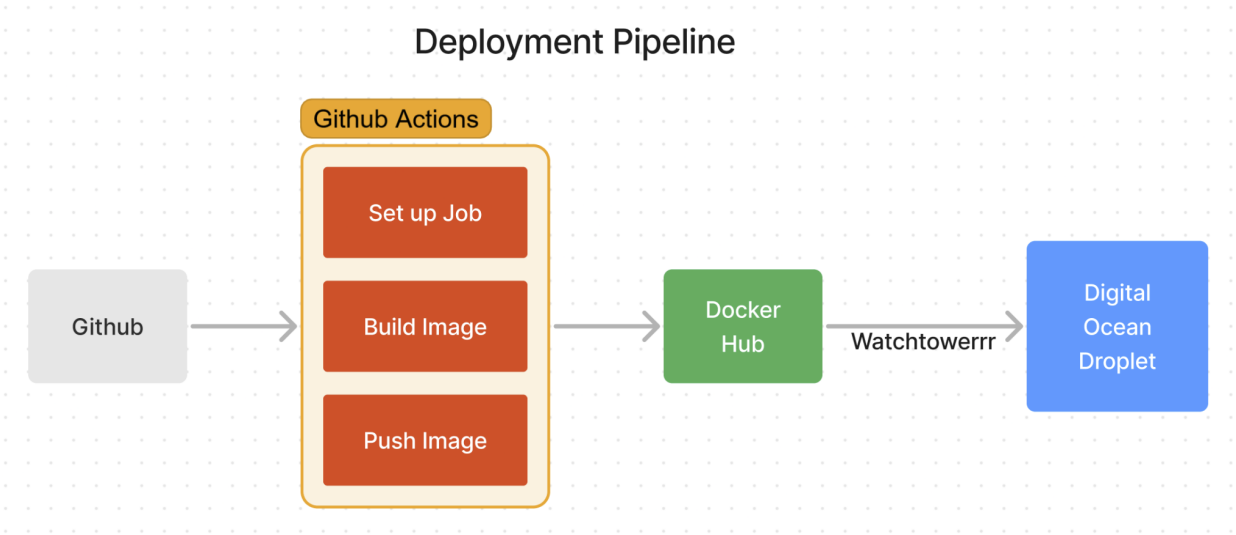
- **Development:** Since this is a personal project, I follow mostly a waterfall approach. I'll create an issue on GitHub, pull it into the project board, and drag it across.
- **Deployment:** When a release is ready, I create a PR to main. This will deploy to production where I can test quickly. If everything looks good, I'll merge to main and the deployment will be official.

System Architecture



System Design

- **Angular 11:** This is the view and front end of the application. Currently, it is responsible for making necessary API calls to the Dynasty Daddy API layer and the Sleeper API layer. Additionally, the front end will generate aggregates, and map responses to data models to use in services throughout the application.
- **Express.js Backend/API:** This component acts as the API layer to fetch data from the Postgres container. In the future, I plan to migrate more responsibility to this level with the translation initiative.
- **Postgres Database:** Maintains all data from KeepTradeCut web scraper.
- **Python Scraper:** This component manages the cron job that is run against KeepTradeCut.com every day. This script uses BeautifulSoup and sleeper_wrapper to tie the create a unique link between the KeepTradeCut players and the Sleeper player ids. Since there isn't a unique identifier for each player, I generate a player id based on fields both responses return and persist those to a linking table in Postgres.



Deployment Pipeline Design

- **Github:** Version Control and Repository Management
- **Github Actions:** Manages the CI/CD pipeline that will build the image and push the image to Docker Hub. The Github actions will be triggered whenever a PR is raised to the main branch or merged into the main branch.
- **Docker Hub:** Container Registry for all built images pushed from Github Actions.
- **Digital Ocean Droplet:** Ubuntu Instance that hosts the production environment from a docker-compose file.
    - **NGINX:** Load balancer for web traffic.
    - **Watchtowerrr:** Monitors Docker Hub for changes and deploys new images automatically every 30 seconds.

Future Timeline

- **Translation Initiative:** Add support for multiple fantasy football platforms (like FleaFlicker). This will increase our potential userbase massively and unlock untapped market potential. This will require large refactors of front-end and back-end infrastructure.
- **API integration migration:** Migrate the Sleeper API responsibility from the front-end angular application to the express backend API layer. This entails moving much of the reactive API calls to the back end and will clean up the client-side networking preventing the client from making dozen of API calls. This will also enable a refactor of Sleeper models to a universal front-end league model to support multiple platform translations in the future.

Glossary

F

- FleaFlicker: Fantasy Football Platform

K

- KeepTradeCut: Crowdsourced Dynasty Rankings for players

S

- Sleeper: Fantasy Football Platform