Software Design Document

Dynasty Daddy

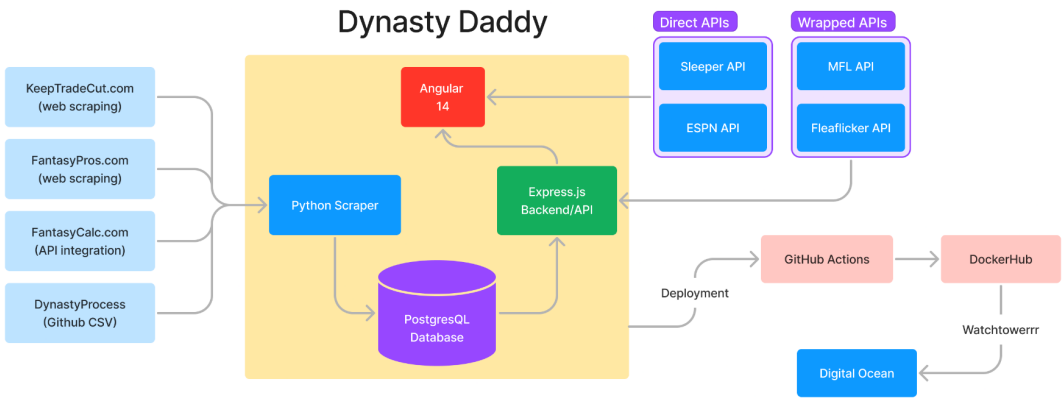
Date: May 19th, 2022

Written by: Jeremy Timperio

Introduction

Dynasty Daddy is a free all-in-one fantasy football tool. It provides users with a frictionless way to find player values, trade calculations, season simulations, league statistics, and more. Utilizing popular fantasy rankings (like KeepTradeCut, FantasyCalc, FantasyPros, and more), Dynasty Daddy ties your league data to player values to power over a dozen tools to help you get an edge in fantasy. 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 platform username to pull league-specific data. In order to do this, I needed to integrate fantasy football platform APIs into my application to pull specific fantasy league data. Another important external requirement was to be able to pull up-to-date player trade values from fantasy market sites. These sites use advanced statistical models to build a table of players and their current trade value. Since many of these sites are smaller projects as well, I introduced a Python cron layer that will handle scraping, integrating with APIs, or parsing CSVs to document a player's trade value on a daily basis. These data points act as the core for our player trade value time series data. By building materialized views off of this database (also handled by the cron job), Dynasty Daddy will have player trend data across multiple fantasy markets and fantasy platforms daily.

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. Sleeper also provides Dynasty Daddy with player stats and projections.
    - Note: It looks like Sleeper has changed to use Graphql going forward so the v1 APIs are most likely legacy.
- **KeepTradeCut Scraping:** 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.
- **MFL API:** MFL has strict throttling protection around their API so I need to be considerate when testing integrations. The Dynasty Daddy API wraps calls to MFL since they lack proper Cors policy.
- **Fleaflicker API:** The Dynasty Daddy API wraps calls to Fleaflicker since they lack proper Cors policy.

General Constraints

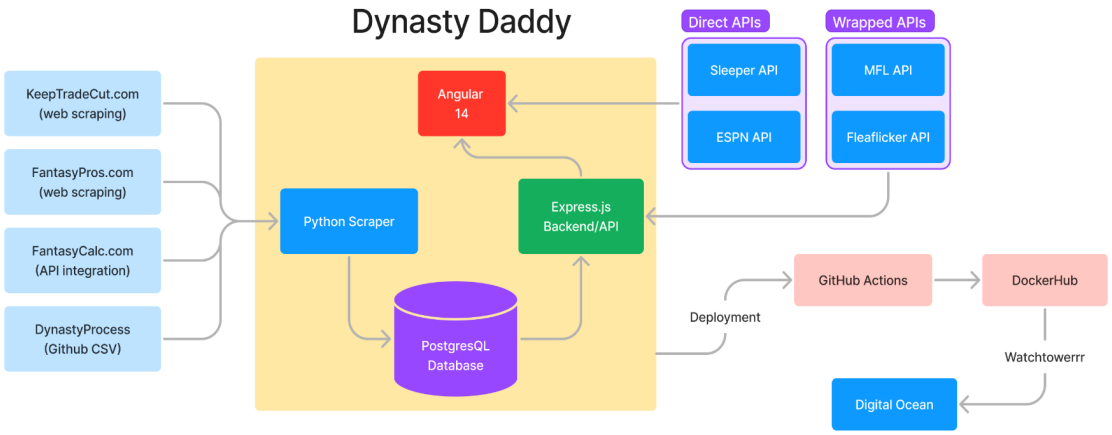
- **Developers:** The largest external constraint is there is only one developer currently, me. Work comes in waves based on how demanding my job, life responsibilities, and motivation for this project is.
- **Money:** Financial backing limits many architectural decisions. As a way to save money, there 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.

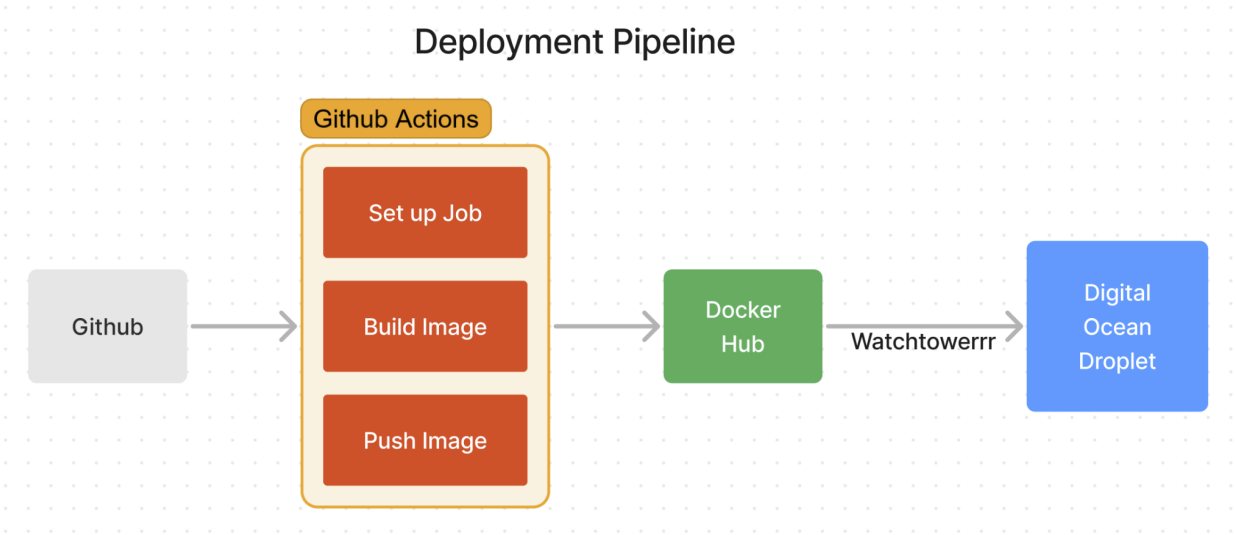
- **Deployment:** When a release is ready, I create a PR to the main branch. Once I test it locally and merge the PR, a build will be kicked off. Within 5 minutes, the code changes will be live in production for users to take advantage of.

System Architecture



System Design

- **Angular 14:** 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 collects all fantasy market data, Fantasy Pros ADP, and player data from Sleeper. After fetching the data, the cron job will transform the data using our internal nameId format and persist the objects to the database. Then once the data has been inserted, it will refresh all the materialized views to update the data being retrieved from the Dynasty Daddy API.



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 Yahoo). This will increase our potential user base massively and unlock untapped market potential. This will require large refactors of front-end and back-end infrastructure.
- **Machine Learning & Modeling:** Dynasty Daddy's database continues to grow with data. This means we can leverage that data using ML to build our own buy/sell and fantasy market predictions. While this is a big initiative, it can really differentiate Dynasty Daddy from the rest of the ranking sites.
- **Refactoring Integration Layer:** Building a more consistent set of best practices for adding new integrations will clean up a lot of the legacy sleeper design idealogy that is currently impacting tech debt.
- **Weekly Scraping for Stats:** Pulling in-house fantasy football stats will improve SEO and Google Lighthouse analytics as well as reduce the network response payload on start.

Glossary

D

- **Dynasty Process**: A fantasy market that utilizes ADP to build player trade values. It will update on a somewhat weekly basis. https://github.com/DynastyProcess

E

- **ESPN Fantasy Football**: One of the most popular fantasy football platforms in the world. It tends to lean more toward redraft leagues and thus has limitations when working with Dynasty Daddy.

F

- **FantasyCalc**: A crowdsourced fantasy market that builds trade values from
- **Fantasy Market**: A term Dynasty Daddy uses to reference a site that creates fantasy trade values. These are sites like KeepTradeCut, FantasyCalc, and DynastyProcess. Dynasty Daddy scrapes fantasy markets daily to build the trend data found on the site.
- **Fantasy Pros**: A fantasy football resource that Dynasty Daddy scrapes BestBall ADPs from in order to power the starter rank and ADP tools.
- **FleaFlicker**: A popular fantasy football platform that leans towards dynasty fantasy football leagues.

K

- **KeepTradeCut**: A crowdsourced fantasy market site that asks users to Keep, Trade, and Cut three players. These data points are used to create a trade value using advanced Elo calculations. This is a very popular fantasy market in dynasty. KeepTradeCut.com

M

- **MyFantasyLeague (MFL)**: A popular fantasy football platform that leans towards dynasty fantasy football leagues. MFL limits API requests and can throttle users if too many requests are made.

O

- **OverTheCap**: A football site that houses NFL players' contracts that is scraped and persisted into the player metadata table. Since this data don’t change often, it can be updated using a manual script.

P

- **Player Profiler**: A football site that houses NFL players' physical profiles and more. This can be scraped and persisted into the player metadata table from a manual script python script. Since this data doesn’t change often there is no need to run it daily.

R

- **Relative Athlete Score (RAS)**: A player prospect metric created by ras.football.com. This score determines how much of an outlier a specific player is from his class and his

position group. Using a CSV, Dynasty Daddy will persist player RAS scores to the player metadata table from a manual script.

S

- **Sleeper**: A popular fantasy football platform that leans towards dynasty fantasy football leagues. Sleeper was the first integration and has the largest user base on the site.