Trade Helper: Fantasy Sports Trade Analyzer

Team Members: Michael Davis, Stephen Kain, Vance Keesler, Matthew Sheehan

Course: CSC 495 – Cloud Engineering

Milestone 1 – Team Formation & Architecture

Overview

Trade Helper is a fantasy sports trade analysis platform designed to help users evaluate the fairness of trades in fantasy football. Users log in securely and can input players from two sides of a proposed trade. The system calculates a score and grade based on player statistics and projected value, ensuring fair and informed trades.

The system is containerized and deployed on Kubernetes to ensure scalability, modularity, and maintainability. Each service runs as a separate pod with clearly defined roles, allowing the team to iterate and scale the platform efficiently.

The system is designed with cloud-native principles in mind. Each component runs as a containerized microservice within Kubernetes, supporting horizontal scalability and high availability. Persistent data is managed through Kubernetes Persistent Volumes, and external APIs are accessed securely via backend services. This modular design ensures that each component can be independently deployed, tested, and scaled.

Component List

Frontend Pod (React + NGINX): Hosts the user interface where users log in, enter trade details, and view results. Exposed via a NodePort service to make the web app accessible externally.

Backend Pod: Implemented as a multi-container pod, consisting of two containers:

- Main Container (Node.js + Flask): Handles incoming requests, evaluates trades, communicates with the database, and fetches data from external APIs.
- Auth sidecar (Auth0 helper container): Manages token verification, session handling, and secure authentication between the frontend and backend.

Database Pod (MongoDB): Stores user profiles, trade history, and player data. Provides persistent storage through a Persistent Volume Claim.

Data Intake CronJob: Periodically updates player and trade data in the database using information fetched from an external API.

External API: Supplies up-to-date player statistics, projections, and injury information used in trade evaluation.

Cloud Infrastructure (Kubernetes): Orchestrates deployment, scaling, and networking for all components. Manages containerized pods and ensures communication between services.

Service Types

NodePort: Used for the frontend to allow the user to access the GUI of the application

ClusterIP: Used for the frontend, backend, trade evaluator, database, and data intake to allow them to all communicate with each other internally, without allowing external access.

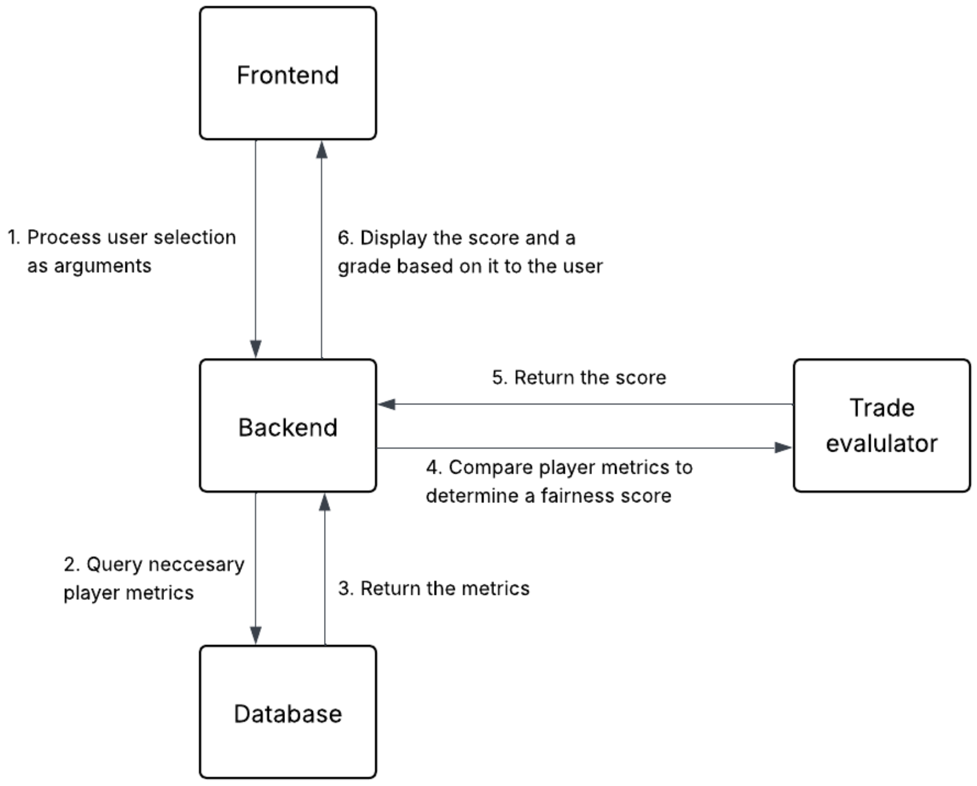
Scaling Target

It is intended that the backend pod can be scaled horizontally. To start, the application will only support scoring trades for Fantasy Football. However, it can be expanded to support numerous different sports, by communicating with databases and trade evaluators for each sport. The backend functions as the system's inference stage and is the primary target for horizontal scaling as user traffic increases.

User Stories

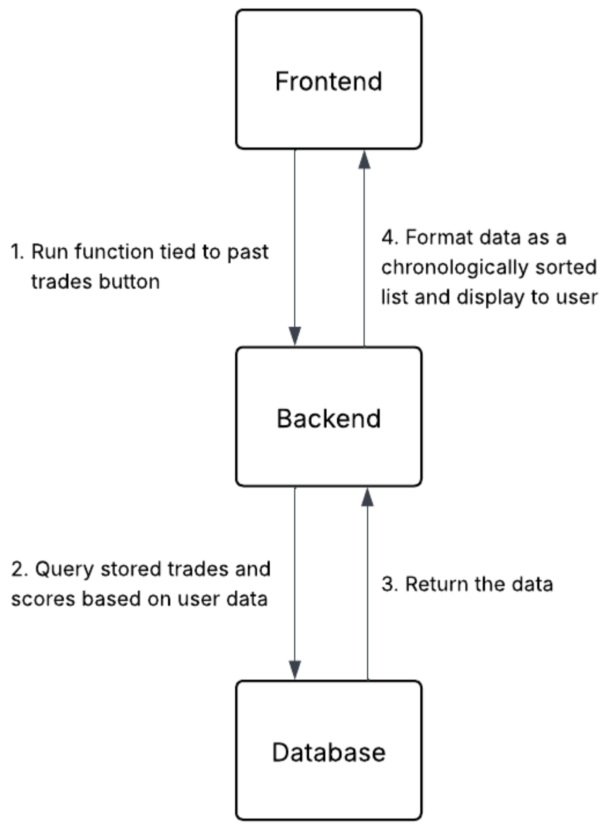
Story: As a logged-in user, I want to check the fairness of a trade so that I can be more educated when trading players in Fantasy Football.

Acceptance Criteria: Given that I am logged-in, when I choose which players I am trading on both sides and press the button to check the trade, then I am given a score and a grade that represents how good that score is.


Frontend
1. Process user selection as arguments
6. Display the score and a grade based on it to the user
Backend
5. Return the score
Trade evalulator
4. Compare player metrics to determine a fairness score
2. Query neccesary player metrics
3. Return the metrics
Database

Story: As a logged-in user, I want to see my previous checked trades so that I can review them whenever I need to.

Acceptance Criteria: Given that I am logged-in and have performed a trade in the past, when I click the button to see past trades, then I am presented with a list of all my past trades and what their scores and grades were.


Frontend
1. Run function tied to past trades button
4. Format data as a chronologically sorted list and display to user
Backend
2. Query stored trades and scores based on user data
3. Return the data
Database

<u>API Contracts</u>

POST /api/evaluateTrade

Request: { “teamA”: [“Jalen Hurts”], “teamB”: [“Patrick Mahomes”] }

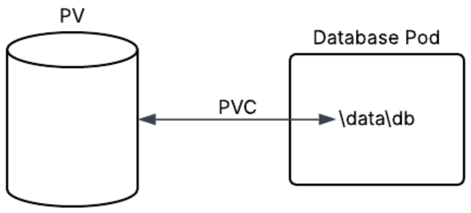
Response: { “score”: 84, “grade”: “B+” }

GET /api/trades

Response: [{“tradeId”:1, “score”:92, “grade”:”A”}, {“tradeId”:2, :”score”:70, “grade”:”C”}]

<u>Persistence Plan</u>

A PV will have to be utilized to store the data contained in the database. This includes the databases for user authentication data, football player metrics, and a user’s saved past trade scores. MongoDB stores all the databases in the \data\db directory, so only one PV will be needed in order to mount this directory into the pod. The database pod will have to utilize a PVC whenever it is attempting to read or write to any of the databases.



<u>Risk & Test Plan</u>

Risks

- External API downtime may lead to stale player data.
- Trade evaluation latency may grow if external stats take too long to fetch.
- MongoDB container restarts could cause temporary unavailability.
- Authentication misconfiguration (Auth0 integration) could block user access.

Test Case	**Input**	**Expected Result**
Happy path	Valid player names, logged in	Returns score and grade
Invalid trade	Missing player name	Returns validation error
Large trade	5+ players per team	Handles gracefully
Data sync	External API down	Falls back to cached stats
Auth	Invalid token	401 Unauthorized

<u>Team Charter</u>

Project Lead / PM – Matthew: Coordinates development progress, manages deadlines, and ensures alignment across all components.

DevOps – Vance: Designs Kubernetes deployment files, manages container orchestration, and oversees CI/CD integration.

Backend Developer – Michael: Implements API endpoints, trade scoring logic, and communication between the backend, database, and external APIs.

Frontend Developer – Vance: Builds and maintains the React user interface, integrates authentication, and connects UI with backend endpoints.

QA / Documentation – Stephen: Tests feature functionality, verifies data flow correctness, and maintains technical documentation for milestones.

Meeting Cadence: Weekly team sync via Discord.

Definition of Done: All pods deployed successfully in Kubernetes and core user stories tested end-to-end.