LevelField: Fantasy Sports Trade Analyzer

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

Course: CSC 495 – Cloud Engineering

Milestone 3 – Final Deliverable

Overview

LevelField 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.

Architecture Recap & Data-Flow Diagram

LevelField is composed of three main pods and supporting cronjobs:

- Frontend Pod: React, Vite, NGINX, served via NodePort on port 30080. Handles user input for trades and displays scores.
- Backend Pod: Multi-container pod including:
  - Main container (Node.js) for trade evaluation, API endpoints, and database communication.
  - Fluent Bit sidecar for centralized logging.
- Database Pod: MongoDB with 1Gi PersistentVolumeClaim mounted at /data/db for storing user data, trade history, and player statistics.
- CronJobs: Periodically fetch and ingest NFL player data into MongoDB.

Data Flow: User submits trade -> frontend sends POST request to backend -> backend evaluates trade using player stats from MongoDB -> backend logs activity to sidecar -> response returned to frontend -> trade persisted in MongoDB.

Pod Grouping & Sidecar Choices

Backend pod uses a sidecar container (Fluent Bit) which captures logs from the main container via a shared emptyDir volume. This ensures logs are centralized and can be exported or monitored without modifying the main application.

All pods are labeled for service selection: app: frontend, app: backend, app: mongo.

Services & Routing

Frontend: NodePort, accessible externally on port 30080.

Backend: ClusterIP, only accessible internally by frontend and cronjobs.

MongoDB: ClusterIP, internal only.

This configuration ensures that user-facing traffic reaches the frontend safely, while internal communications and database access are restricted to the cluster.

Scaling & Self-Healing Results

Backend Deployment: Configured for horizontal scaling (replicas adjustable).

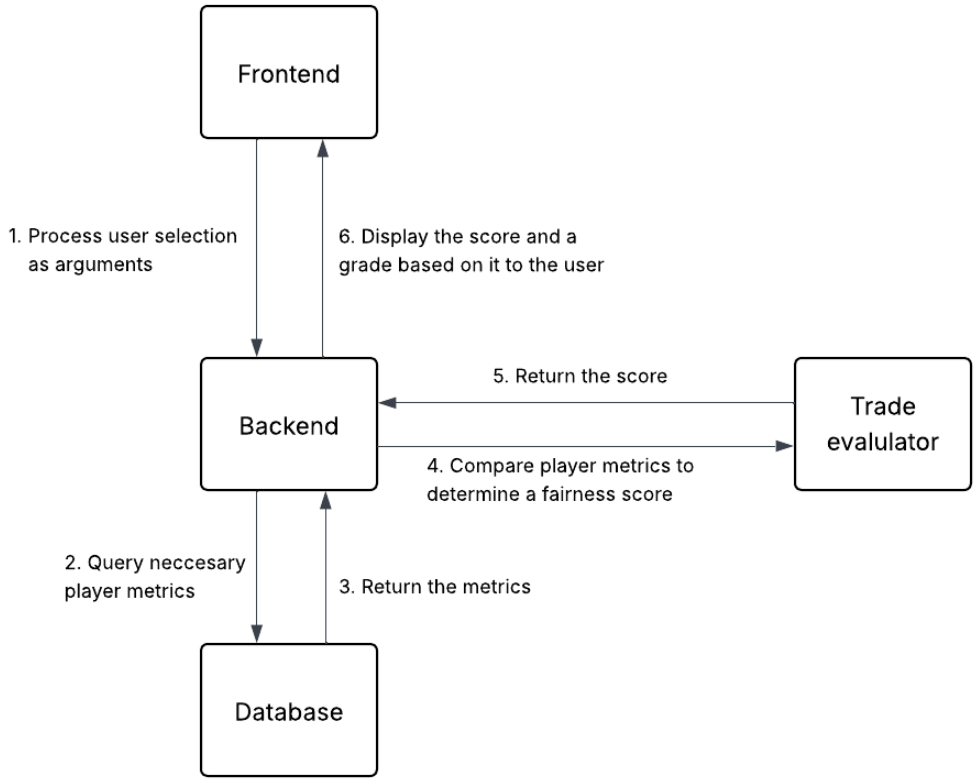
Pods: Kubernetes automatically restarts pods on failure and monitors readiness/health.

Sidecar: Logs remain persistent across container restarts via emptyDir volume (lost if pod is deletedbut captured in Fluent Bit output).

User Stories

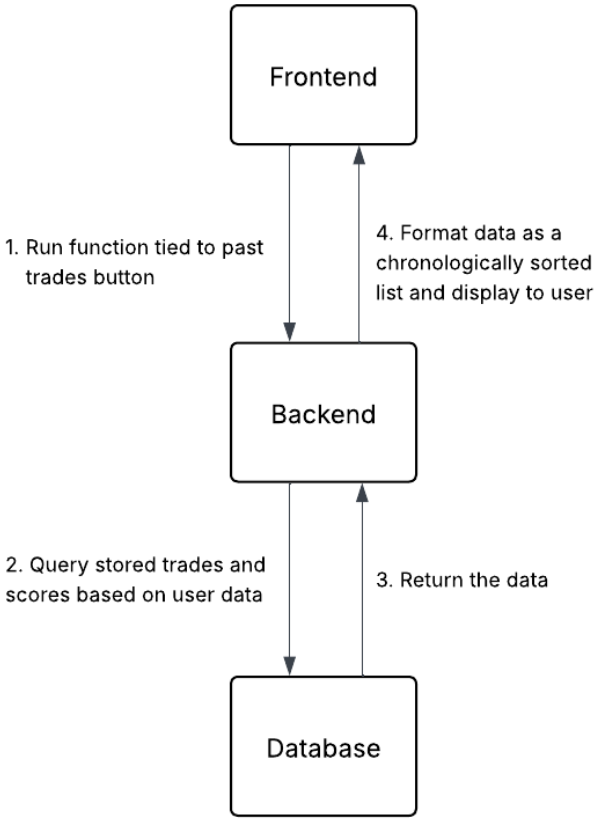
Story: As a logged-in user, I want to check the fairness of a trade so that I can be more educated when trading players in Fantasy Football.

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.



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.



API Contracts

POST /login

Initiates login sequence handled by auth0

POST /trade

Request: { “sentplayers”: [“Jalen Hurts”], “receivedplayers”: [“Patrick Mahomes”] }

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

GET /previoustrades

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

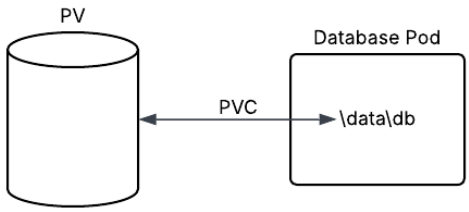
Persistence Plan & Verification Steps

PersistentVolume: MongoDB stores all data at /data/db on a 1Gi hostPath volume mounted via PVC.

Verification Steps Taken:

1. Confirmed MongoDB pod mounts PVC correctly and can write/read data.
2. Created log entries in /var/log/app/app.log inside the backend pod to verify Fluent Bit sidecar captured log lines in stdout.

Limitations: EmptyDir volumes are ephemeral and do not persist if the pod is deleted.



Risk & Test Plan

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

Known Limitations & Future Work

Known Limitations:

- Due to runtime environment and CORS issues, frontend currently cannot communicate with backend. Resolving this will require rebuilding frontend to respect runtime API URLs.
- MongoDB is single-node; no high availability or replication
- Rancher Desktop single-node setup limits testing of true cluster behavior
- Paywalled out of pulling from current season datasets that would allow more accurate trade evaluation

Future Work:

- Complete backend-frontend integration for trade evaluation
- Implement HA MongoDB with replication and backups
- Improve CronJob scheduling for error handling
- Add support for additional sports
- Introduce proper ingress and TLS for external frontend access
- Introduce ingress controller for more control over entry point and user routing

- Transition to the Sleeper API which provides free, up-to-date fantasy football data, reducing dependency on external paid datasets