EECS 581 - Project 3: Fanalytics

Team 27 - System Architecture Document

Asa Maker, Brandon Dodge, Zach Sevart, Josh Dwoskin, and Ebraheem AlAamer

October 23, 2025

Synopsis

Fanalytics is a cross-platform sports analytics app for visualizing, comparing, and analyzing player and team performance using real-time data, advanced metrics, and scalable modular architecture.

1 System Architecture Overview

1.1 Purpose

The purpose of this architecture is to define a scalable and maintainable system for real-time and historical sports analytics. Fanalytics is designed for coaches, analysts, and fans who want to interpret player and team data visually and statistically. The architecture enforces separation of concerns between the data ingestion pipeline, analytics engine, and user interface. This separation ensures accurate computations, secure data handling, and efficient rendering of dashboards, enabling each module to be tested and extended independently.

The design uses a clear data flow: user interaction or scheduled ingestion triggers API calls → analytics processing → database updates → visualization. Explicit system states (Idle, Processing, Ready, Error) allow predictable transitions and debugging. Figure 1 shows the major pathways between components.

1.2 Components

This architecture consists of six primary components:

- **User Interface (Web & Mobile):** Developed in React (web) and React Native (mobile), the UI displays interactive dashboards, charts, and reports. It manages authentication, navigation, and visualization using data fetched from backend APIs. Controls include login, upload CSV, select team, compare players, and export results.
- **API Gateway (Node.js/Express):** Implements RESTful endpoints for CRUD operations on player, team, and match data. Handles JWT authentication, rate limiting, and routing between client and backend services.
- **Ingestion Service:** Retrieves external data from APIs such as SportsRadar, parses CSV uploads, and standardizes information into a canonical schema before persisting it in the database. Operates on scheduled intervals or manual triggers.
- **Analytics Engine (Python/Flask):** Processes ingested data to compute advanced metrics (e.g., player efficiency ratings, expected goals, win probabilities). Implements machine learning models for projections. Interfaces with the API Gateway via HTTP or message queues.
- **Database Layer (MongoDB & Redis):** Stores structured data such as users, teams, players, and metrics. MongoDB Atlas manages persistent data; Redis provides high-speed caching for frequently accessed dashboards.
- **Cloud Infrastructure (AWS):** Hosts backend services using EC2, manages storage with S3, and triggers ingestion jobs through Lambda. All services communicate securely via HTTPS and use IAM roles for role-based access control.

1.3 Data Flow

- **User Request:** A user logs in, selects a team or player, and submits a query.
- **Frontend Action:** The React interface validates inputs and sends API requests.
- **API Gateway:** Validates the request and routes it to either the Analytics Engine or Database Layer.
- **Ingestion & Analytics:** If new data is required, the ingestion service fetches and normalizes it. The analytics engine computes updated metrics.
- **Database Update:** New or recalculated data is written to MongoDB; cached summaries are refreshed in Redis.
- **UI Response:** The backend sends JSON responses that the UI visualizes in charts or tables.

Figure 2 visualizes this end-to-end flow from user to visualization.

1.4 Key Data Structures

Player: player_id, name, team_id, position, stats (dictionary of key metrics), updated_at.
Team: team_id, name, roster (list of player_ids), record, average_metrics.
Game: game_id, team_A_id, team_B_id, scores, play_events, date.
Metric: metric_id, entity_id (player/team), season, value, source, timestamp.
User: user_id, email, role (admin/analyst/viewer), preferences.

These structures are normalized for efficiency but maintain flexibility for schema evolution as more sports are added.

1.5 Assumptions & Constraints

Assumptions:

- Users have stable internet access and modern browsers or mobile devices.
- Data providers offer consistent, documented APIs.
- Real-time analytics refresh intervals are acceptable at 5–10 minutes of latency.
- Users include admins, analysts, and general viewers, each with predefined permissions.

Constraints:

- Strict privacy rules prevent sharing user credentials or third-party API keys.
- All APIs must use HTTPS.
- Processing large CSV uploads (>100MB) is throttled to prevent service overload.
- Deployment must use AWS for hosting, S3 for file storage, and MongoDB Atlas for persistence.

1.6 Extension Points

- **AI Predictions:** Integrate deep learning models for player injury risk or win probability forecasts.
- **Social Integration:** Allow sharing of performance dashboards to social media platforms.
- **Custom Metrics:** Permit analysts to define custom formulas for metrics within the UI.
- **Enhanced Visuals:** Introduce real-time 3D visualizations for plays and heatmaps.
- **Multi-Sport Expansion:** Extend architecture to football, soccer, and baseball through modular schema additions.

2 System Diagrams

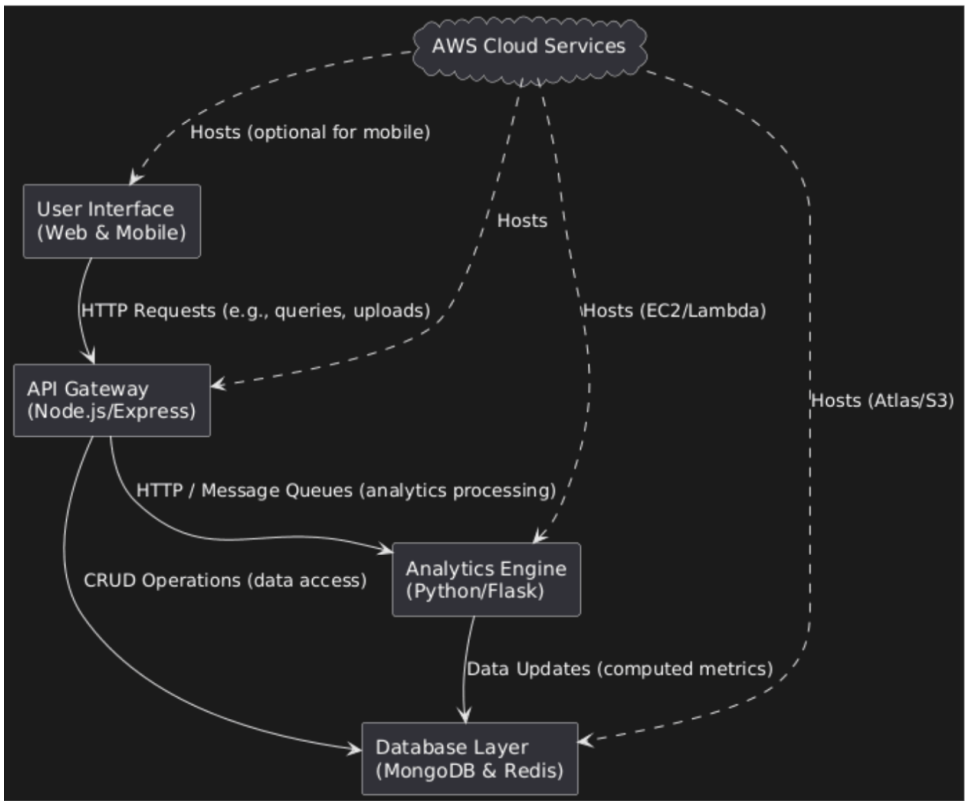

Figure 1: Component Interaction Diagram - shows User Interface, API Gateway, Analytics Engine, Database Layer, and AWS cloud services interconnected.

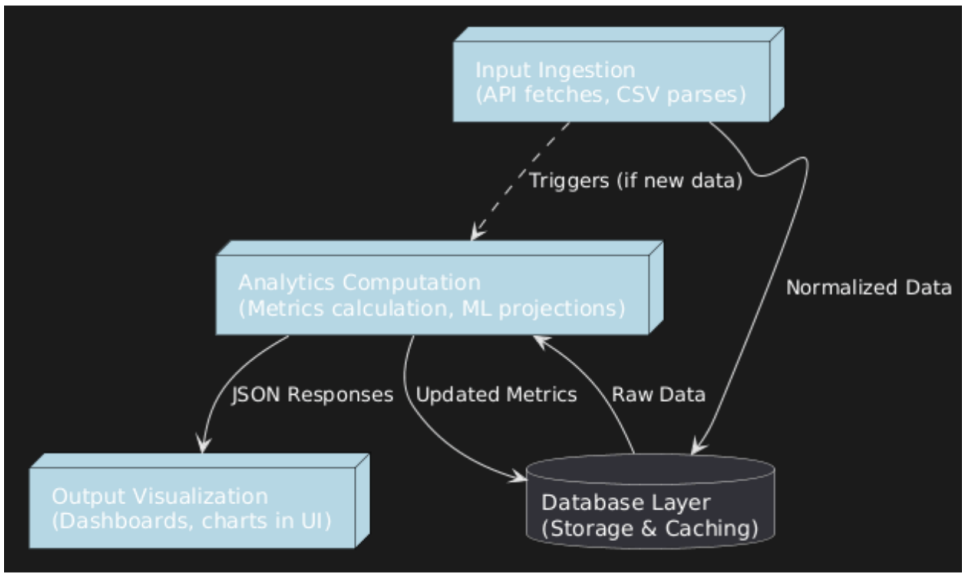

Figure 2: Data Flow Diagram - illustrates input ingestion, analytics computation, and output visualization.

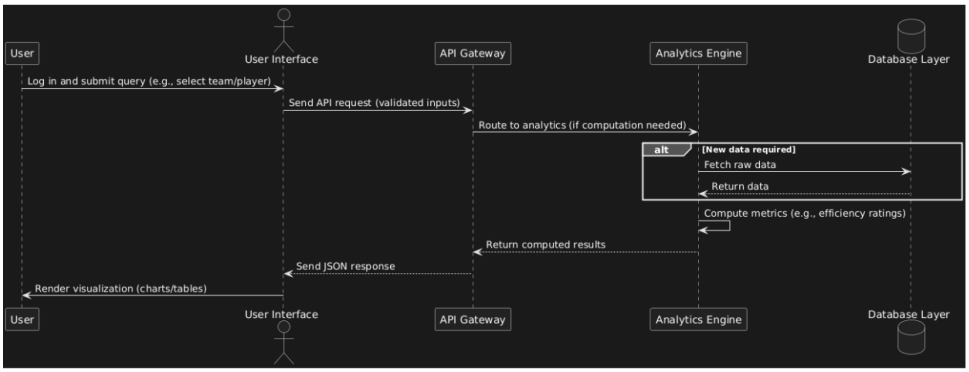

Figure 3: Sequence Diagram - depicts user querying a dashboard through API request and backend computation.

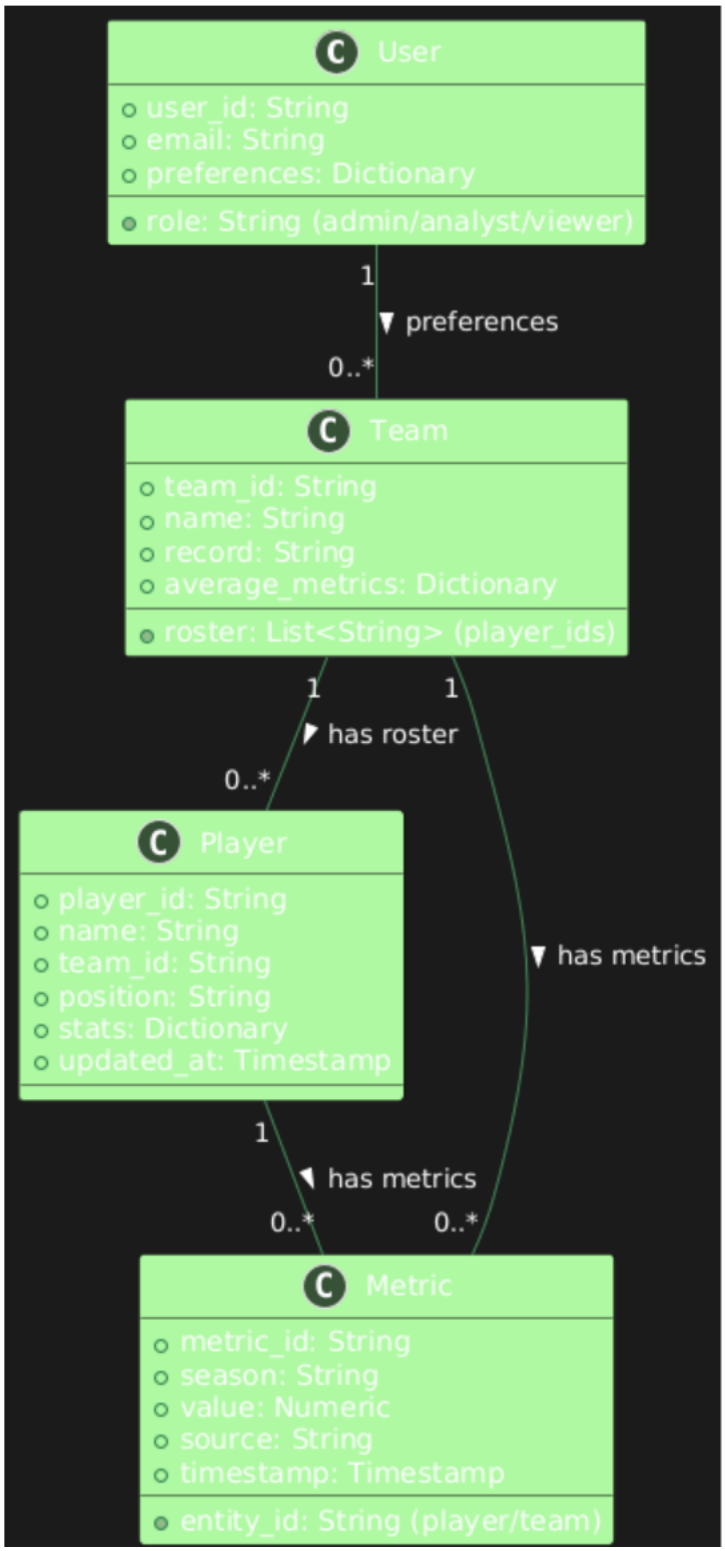

Figure 4: Database Schema Overview - shows entity relationships among Users, Teams, Players, and Metrics.