IT3041 - Information Retrieval & Web Analytics

Sri Lanka Institute of Information Technology

ATHLETE – Multi Agent Football Analysis Platform

Final Report

Team SYNCODE

Website Link : https://athlete-analysis.vercel.app/

GitHub Repository : https://github.com/SYNCODE-SLIIT/Sports-Analysis

Product Video : IRWA Final Video

Date of submission : 31st October 2025

Submitted By :

IT Number	Student Name	Student Email Address	Contact Number
IT23439078	Perera E V	it23439078@my.sliit.lk	+94 77 239 7767
IT23157132	Jayakody J A S R	it23157132@my.sliit.lk	+94 71 484 1673
IT23148086	Perera P D	it23148086@my.sliit.lk	+94 77 154 6769
IT23141506	Liyanage L N P	it23141506@my.sliit.lk	+94 78 706 9741

Abstract

This document created as the final report for the IT3041 – Information Retrieval & Web Analytics module presents our team's project ATHLETE, an AI-powered multi-agent sports analysis platform which was developed as a comprehensive solution for sports enthusiasts giving them personalized sports insights and recommendations. The system uses a modular agent-based structure, integrating multiple sources of data and advanced AI models to give the users a smooth experience. The system delivers real-time match insights, analytics, summaries, highlights & tailored news for each user. Key components include collector, summary, highlight and analysis agents, chatbot, and a recommendation system which co-ordinate with each other to process and enrich sports data efficiently.

The platform uses modern web technologies (Next.js, React, FastAPI) and robust backend services (Supabase, Stripe) to ensure scalability, security and seamless user experience. Responsible AI practices were used throughout the system to address fairness, transparency and data privacy. This document also highlights the commercialization strategy for targeting sports enthusiasts and professional analysts with a focus on sustainable business models. Evaluation & Results demonstrate the accuracy of the system to effectively deliver timely and relevant match information. This also highlights areas for future enhancement in to enhance user experience.

In summary this report serves as a comprehensive reference to all aspects of the ATHLETE platform including technical, ethical and commercial aspects of the implemented platform.

Table of Contents

1. Introduction

1.1 Background & Motivation

ATHLETE Multi-Agent Sports Analysis System is an intelligent, agentic AI-based framework, the system aims to enhance real-time sports analytics by forms of automation, collaboration, and explainable data insights. The system integrates multiple specialized agents – Collector, Summary, Highlight, and Analysis Agents. These agents co-operate to collect, interpret and analyze live and past match data from two main external APIs TheSportsDB and AllSportsAPI.

In the current field of Information Retrieval(IR), the sports domain is a broad and an ideal application area, In this field football or soccer is the most famous sport there is. Massive volumes of structured and unstructured data are generated every second from football matches around the globe. This data must be efficiently retrieved, processed, analyzed, and summarized to gain valuable insights from them. Traditional single-model systems struggle when managing such dynamic and high-frequency data streams. This requires scalable AI-driven pipelines and by employing a multi-agent framework with Natural Language Processing(NLP) & and Large Language Models(LLM), this project has demonstrated how these components can be combined to enable automated reason, context-aware summarization and predictive analysis in sports.

The motivation behind this system is to provide fans with real-time and personalized analytics and match insights and to help professionals analyze their teams all through our platform

1.2 Problem Statement

In modern sports analytics a massive amount of data is continuously generated through live match feeds from multiple sources. Despite this abundance many existing analytical systems face major limitations in automation, interpretability and data integration. Most of these systems are dependent on fragmented workflows and can result in inconsistent data , not being available in real-time and limitations in analytical capability. Furthermore, conventional tools are primarily focused on static performance summaries, they lack autonomous decision making and don’t use agentic collaboration for interoperable communication and task distribution.

Sports enthusiasts face many limitations when using current sports analytics platforms,

- **Fragmented Data Sources –** Most users rely on multiple websites or APIs this leads to incomplete or inconsistent information
- **Lack of Real-time Analytics –** Most websites offer post-match summaries rather than real-time live insights
- **Limited Explainability –** Users receive results without understanding the reasoning behind them
- **Security & Reliability Concerns –** Weak API authentication, rate limits and poor error handling exposes uses to data risks.

"Existing sports analytics systems don't provide unified, real-time, and explainable match insights due to fragmented data sources and limited automation, Thus an intelligent multi-agent system that can automate the collection, analysis and interpretation of raw sports data with transparency & accuracy is required"

1.3 Objectives

The primary objective of this project is to design & implement an Agentic AI-based Multi-Agent Sports Analysis System that can collect, process, analyze live sports data intelligently to generate meaningful and transparent insights

Specific Objectives

- **Develop a Multi-agent Architecture** consisting of Collector, Summary, Highlight and Analysis agents that communicate and work collaboratively to automate data collection and analysis
- **Integrate Information Retrieval and NLP Techniques** to extract, summarize, and interpret match details from 2 main sports data sources
- **Provide Real-time Analytical Insights,** including win probability estimation based on past results, match and league summaries and match-level predictions
- **Ensure Explainability and Transparency** in all analytical outputs through which follow responsible AI practices
- **Implement Data Privacy and Security Measures** such as authentication, input sanitization and encrypted communication between agents
- **Support scalability and commercialization,** through a web-based deployment

2. System Design

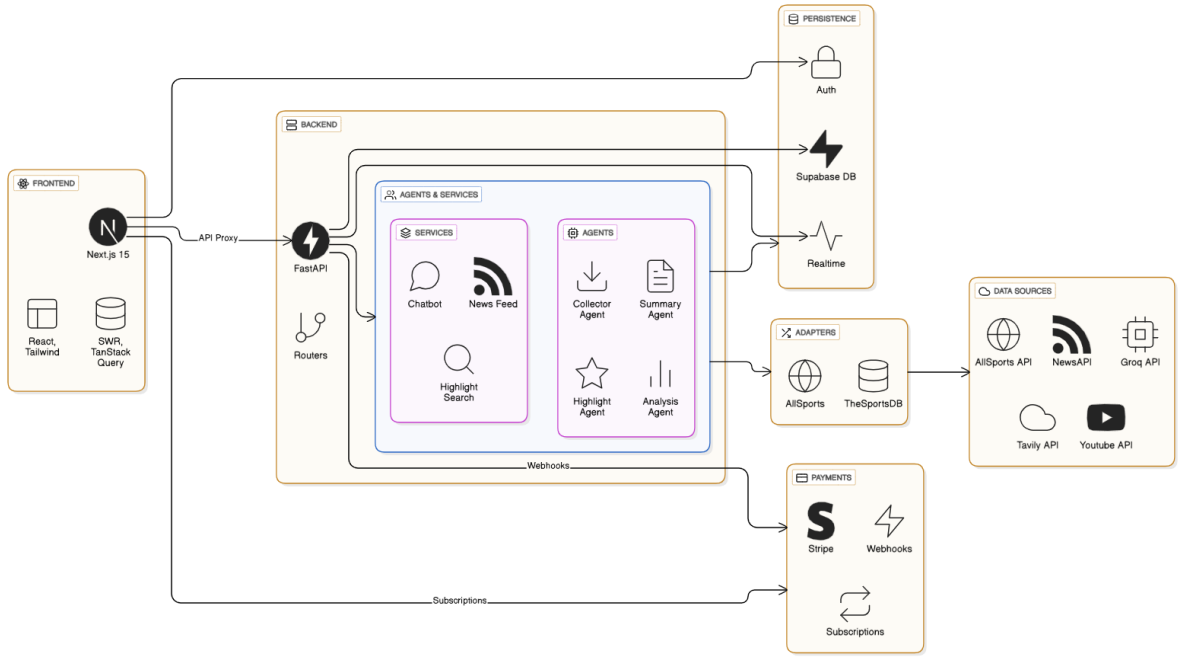
2.1 Overall System Architecture

The ATHLETE platform is organized as a three-layer, agent driven architecture where ingestion and processing occur in modular backend workers, persistent artifacts and metadata are stored in object and document stores, and presentation is handled by Vercel hosted React/Next.js frontend that consumes a documented REST surface.

Ingestion begins with the collector agent accepting API data, detected events and timestamps are stored in an event log and a metadata catalog that serve as the canonical source of truth. The summery agent consumes the event log to produce structured textual summaries, the Analysis agent computes numeric metrics and visual artifacts, and the Highlight agent uses event timestamps to clip, transcode, and publish web optimized highlight clips, and highlight agent extract special events and create timeline including those events.

A central backend API server exposes endpoints for match listings, summaries, highlight indices, and analytics, issues preassigned or CDN URLs for media delivery, and coordinates admin operations such as user management. External integrations include Stripe for payments, Groq for AI models, and Tavily for web search in the chatbot. This setup ensures scalable, real-time processing from data ingestion to user delivery.

2.2 Architecture Diagram



The frontend acts as the entry point, proxying request to avoid CORS issues and ensuring seamless data fetching. Backend orchestration via FastAPI's routers dispatches intents to agents, which abstract complexities like API fallbacks. Adapters synchronization for live updates. Stripe integrates at the persistence level for monetization. This architecture support horizontal scaling, with agents deployable independently if needed.

2.3 Methodology

ATHLETE platform's methodology centers on a multi-agent AI architecture for real-time football data processing, integrating, machine learning and generative AI to deliver insights. It seems likely that this approach balances data accuracy with user engagement, though dependencies on external APIs introduce some variability in performance.

- **Core Algorithms:** Machine learning models analyze team performance, player stats, and historical data for predictions like win probabilities, with real-time adjustments based on game state.
- **Primary Tools:** Data integration from APIs like AllSports and TheSportsDB, AI models from Groq for summarization and analysis, and Supabase for persistence.
- **Key Techniques:** Real-time data aggregation, fallback mechanism for reliability, and AI orchestration for personalized insights, emphasizing informational outputs over guarantees.

The core algorithms focus on data processing, prediction, and content generation, tailored for football-specific analytics.

- **Win Probability Calculation**: This algorithm uses a heuristic-based model to estimate match outcomes in real-time. It factors in current score, remaining time, team form, head-to-head records, and player performance metric. For instance, the model might assign weights to variable like home advantages and adjust dynamically. Updates occur via live data feeds, ensuring probabilities reflect game events like goals or substitutions. This is implemented in the Analysis agent.
- **Timeline and Highlight Generation**: In the Highlight Agent, algorithms normalize event data (goals, cards, subs) into chronological timelines. This involves parsing

algorithms for minute extraction and type classification. Video aggregation uses scoring algorithms to rank candidates by relevance, with sorting by timestamps.

- **Summarization and Narrative Generation**: Leveraging large language models (LLMs), this algorithm crafts Markdown summaries from raw match data and summaries of new articles related to matches and leagues. Prompts guide the AI to focus on key events, and outputs are structured as bullet points and paragraphs, ensuring brevity and accuracy.

The platform integrates a suit of tools for data handling, AI, and user interaction, selected for their reliability and compatibility.

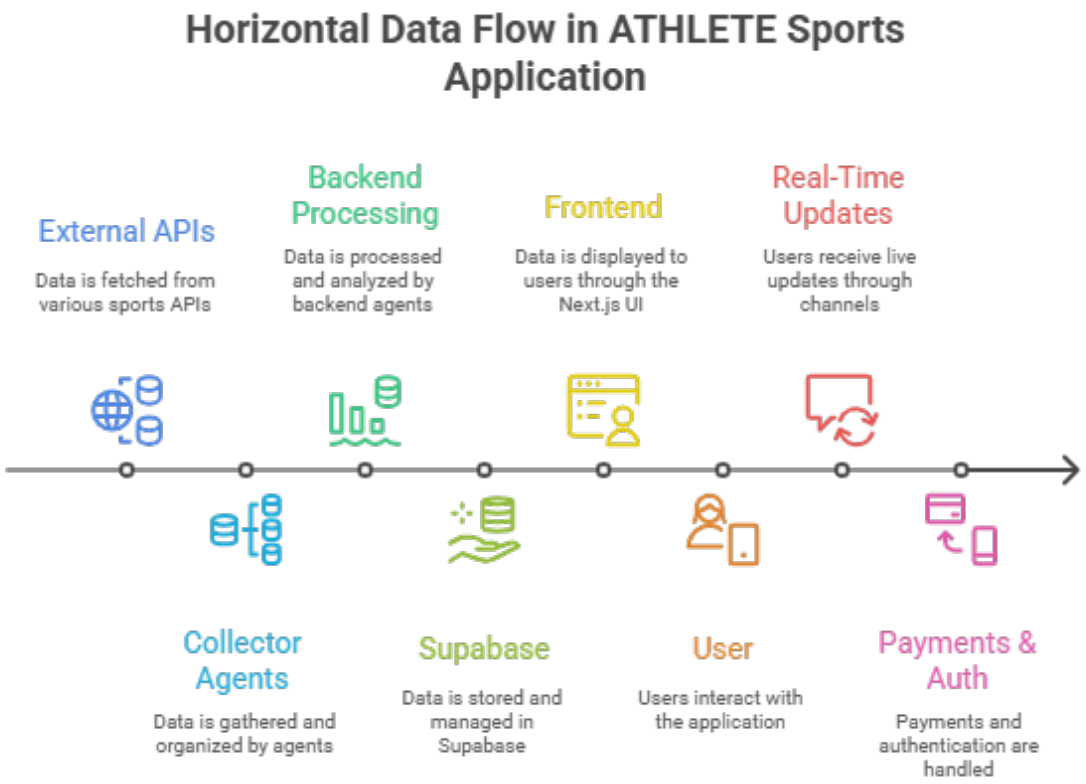
- **External APIs**: AllSports API and TheSportsDB, NewsAPI these tools enable data ingestion, with adapters abstracting differences in payloads.
- **AI Models and Services**: Groq provides LLM inference with models like llama3. Tavily supports web searches in the chatbot, enhancing query responses. YouTube API fetches video highlights, with requests library for HTTP calls.
- **Database and Persistence Tools**: Supabase handles PostgreSQL storage, authentication, real-time channels, and RPCs. It stores user preferences, match data, and subscriptions, using SQL migrations for schema management.
- **Payment Integration:** Stripe manages subscriptions, trial (7-days free), and billing portals via webhooks and Typescript libraries.
- **Testing and Quality Tools:** Pytest for backend unit tests, npm lint for frontend code quality.

Techniques emphasize robustness, real-time capabilities, and ethical AI practices.

- **Multi-Agent Architecture:** Agents divide tasks, Collector aggregates data with fallbacks, Summary uses LLM prompts for narratives, Highlight applies tagging and deduplication for timelines, Analysis compute metrics. Inter-agent communication via internal calls and shared payloads promotes modularity.
- **AI Orchestration and Prompt Engineering:** For Summarization and chatbot, techniques involve structured prompts and model chaining. Bias mitigation draws from diverse sources, with disclaimers for outputs.

- **Video and Search Techniques:** YouTube searches use query building with time windows (days around kickoff) and filtering (team terms in titles/descriptions), Scoring ranks by relevance and recency.
- **Security and Privacy Techniques:** Supabase row level security and authentication, Stripe for compliment payments. Ethical techniques include non-advisory disclaimers and admin controls for monitoring.

2.4 Workflow



Step-by-Step Data Flow

1. **Initiation**: User accesses the web app, triggering API requests for match data.
2. **Collection**: Backend agents fetch and merge data from primary and fallback sources.
3. **Enrichment**: AI processes raw data into summaries, highlights, and probabilities.
4. **Storage and Sync**: Processed data is cached and updated in real-time via databases.
5. **Rendering**: Frontend displays insights, with interactive elements for Pro users.

2.5 Technological Stack

- **Frontend:** Next.js, React, Tailwind CSS, Radix UI, TanStack Query.
- **Backend:** FastAPI (Python), with custom agents and services.
- **Database and Auth:** Supabase (Postgres, Auth, RPC).
- **Payments:** Stripe for subscriptions, trials, and billing portals.
- **AI and Search:** Groq models.
- **APIs:** AllSports, TheSportsDB, NewsAPI, Youtube Data API.
- **Others:** Pytest for testing, npm lint for frontend quality.

3. Agent Design & Implementation

3.1 Collector Agent

The Collector Agent is the backbone of the of the Athlete system, performing all the data retrieval tasks from the main sports APIs. It is responsible for orchestrating all queries related to related to matches, leagues, teams, players etc. Its main purpose is to retrieve data from multiple APIs and present it to the front end and the other agents using a single interface.

Purpose & Functionality

- **Multi-Provider Unification**: The agent uses two main data sources. **AllSports** (the primary provider for most live data) and **TheSportsDB** (used as a fallback and for specific supplemental data).
- **Intent-based routing:** The agent operates on a simple **intent-based system**. The frontend or other agents pass intents (e.g., events.list, league.table, player.honours) to the router endpoint **/collect** with arguments and this router decides which API to use and checks for errors and then retrieves the data.
- **Broad Capability Coverage:** This dual-provider strategy provides comprehensive data coverage. **AllSports** is used for dynamic data like live scores, event details, league standings, odds, and head-to-head (H2H) statistics. **TheSportsDB** supplements this with more static or specialized data, such as venue information, TV listings, and detailed player histories (e.g., honours, former teams, contracts).
- **Intelligent Fallback Logic:** If the primary API (**AllSports**) return and empty payload, the router automatically re send the request to the second provider (**TheSportsDB**), which ensures that the user receives data whenever possible.
- **Flexible Data Resolution:** To ensure user-friendliness and ease of use by other agents, the collector agent includes robust name-to-ID resolution that converts regular name based requests (e.g. "Manchester United", "Primer league") to ID based requests that can be used to request data from the API.
- **Utility and Analysis Integration:** Beyond simple data fetching, the Collector Agent's router integrates with other agents to provide value added insights. It can dispatch requests to the **AnalysisAgent** to compute win probabilities and form analysis, or to the **HighlightAgent** to assemble highlight reels and timelines and **SummarizerAgent** to generate match summaries.

Implementation Details

The **Collector Agent** operates through a coordinated pipeline spanning the frontend proxy, FastAPI router, and backend adapters, ensuring robust communication and data consistency across all components.

Request Flow

1. **Browser → Next.js API:** The frontend calls **postCollect({ intent, args })** which proxies the request to **/api/collect.**
2. **Next.js / Other Agents → FastAPI:** The proxy forwards the JSON payload to POST /collect while backend agents directly call it, this invokes the **RouterCollector** handler.
3. **RouterCollector → Providers:** The router selects the appropriate data source (**AllSports or TheSportsDB),** fetches the result, applies fallback logic if needed, and returns a standardized response.

Each response follows a unified format:

```
{ "ok": true, "data": {...}, "meta": { "trace": [...], "source": {...} } }
```

RouterCollector

This is the central point that handles incoming requests. It consists of a map that decides which API to use based on the intent. Also consists of the fallback logic which gets triggered by checking if a provider's response is "empty-ish" or has failed. It exposes utility methods (e.g., get_live_and_finished(date)) that aggregate data from multiple intents to construct a complete view for the UI, such as a daily scoreboard.

Utility Methods

- **get_live_and_finished(date)** merges live and completed matches for the same day.
- **get_history() / get_history_dual()** aggregate historical fixtures and compare both provider outputs.
- **meta.source** fields capture which provider produced the data and whether a fallback occurred.

Data Sourcing Agents

- **AllSportsRawAgent**: This is the primary agent that acts as a pass-through client for the **AllSports API.** Selected due to its vastness of data even in the free version. It manages **ALLSPORTS_API_KEY** authentication, in-memory caching (for leagues, countries), and **name-to-ID resolution**. It also provides optional enrichments, such as synthesizing a match timeline from raw event data or computing a simple "best player" heuristic.
- **CollectorAgentV2**: **(TheSportsDB API).** This is a more rule-based, soccer-focused agent. It provides robust name resolution and acts as a fallback if the primary agent doesn't respond.

Configuration & Environment

Key environment variables:

- ALLSPORTS_API_KEY, ALLSPORTS_BASE_URL
- THESPORTSDB_API_KEY
- Local caching TTLs and internal base URLs managed via .env files.

Practical Example

For a request such as:

```
{ "intent": "events.list", "args": { "date": "2025-10-28" } }
```

1. The frontend posts to /api/collect.
2. The proxy forwards to FastAPI /collect.
3. RouterCollector calls AllSports first; if the response is empty, it retries via TheSportsDB.
4. The selected provider's raw event list is returned to the UI with metadata showing which source was used.

This Agent Allows Athlete to dynamically adopt and display up-to-date information and allow other agents to perform by providing all relevant information and data.

3.2 Summary Agent

Purpose & Functionality

The summarizer Agent serves as the central intelligence module of the ATHLETE multi agent architecture. Its main purpose is to generate coherent, factual and human readable summaries of football matches and sports news articles by integrating live match data, analytical context and natural language generation.

It is communicating with collector agent and retrieves structured match data, and this information is processed and transformed into well-structured summary containing a headline a detailed paragraph, bullet points and key events.

The agent supports two summarization modes

- Match summaries-Generate detailed match reports by combining normalized data from multiple data providers. It ensures factual accuracy by grounding all narrative details on real statistics, events and metadata
- News Article Summaries-Fetches and summarizers external sports articles or web content into concise narratives and bullet points using Groq LLM. The summaries are factual, on-speculative and designed for direct embeddings into ATHLETE front end dashboard

The summarizer agent is designed to give accurate, consistent and scalable summaries. It enforces strong factual alignment with provider data, prevents hallucinated content and offers fallbacks to deterministic template generation when LLM access is unavailable. Through its build in error handling, caching and concurrency support, it ensures real time responsiveness for both live and complete matches

Implementation Details

The summary agent is implemented as an asynchronous FastAPI microservice that integrates multiple layers of functionality, data retrieval, normalization and text generation into one pipeline

System configuration and initialization

- uses environment variables for flexible deployment modes (AGENT_MODE)
- Connect via HTTP to collector agent endpoint(/collect)

- Integrated Groq LLM API with configurable parameters (GROQ_MODEL, LLM_TEMPERATURE) for text generation

Input models and validation

- Defines pedantic models (summaryRequest ,SummaryOut, NewsSummaryRequest,) to strictly validate incoming JSON payloads
- Ensures schema enforcement across endpoints to prevent malformed or incomplete data
- Data retrieval and normalization
- The fetch_event_bundle() use provider and collects match specific data
- Handles missing data
- Prompt construction and LLM integration
- The build_llm_prompt() function dynamically constructs structured system and user prompts that include match data, timeline summary,contextual story cues
- These prompts are passed through rull_llm() which invokes Groq model to generate a structured JSON output.
- If LLM response is incomplete, fallback template are used to maintain summary continuity
- News summary Generation
- For /summarizer/news, the agent fetches online articles using HTTPs and beautiful Soup cleans the HTML, extracts readable content and clips it to the character limit.
- Text is then summarized via summarize_news_article(), which provides concise narrative and key bullet points while ensuring copyright safe, non-speculative wording

Concurrency and performance

- The entire agent is built around async/await design principle, enabling multiple simultaneous summarization requests

Outputs

The final output by summarizer agent is a JSON object compatible with the frontend dashboard containing

- A headline and summary paragraph
- 3-6 bullet points summarizing key insights

3.3 Highlight Agent

Highlight Agent is a backend component in the ATHLETE platform, primarily responsible for processing and normalizing match data to create timelines and aggregate video highlights for football matches. It seems likely that this agent enhances user engagement by providing structured, visual recaps of key events, though is effectiveness depends on API availability and data quality.

- Core Role: Builds match timelines from events like goals, cards, and substitutions, and collect video highlights from multiple sources to support frontend display.
- Supported Features: Handles intents for timelines (highlight.timeline) and videos (video.highlights), ensuring consistent data formats for real-time insights.
- Integration: Works within a multi-agent system, relying on adapters for data fetching and services for additional processing, with fallbacks for reliability.
- Limitations and Considerations: Dependent on external APIs like AllSports and YouTube, which may introduce latency or require keys; outputs are informational and not guaranteed for all matches.

Purpose & Functionality

The primary purpose of the highlight agent is to bridge raw data from collector agent with frontend ready formats, enabling the display of matches timelines and video highlights in a unified manner. In the context of the Athlete platform, which targets football enthusiasts with features like live scores, win probabilities, and AI chatbots, the agent addresses the need for engaging, media-rich content. It processes payloads from collector agent, normalizing events into Timeline item objects that capture minutes, teams, types (Goal, yellow card…), players, assists, and notes. This supports timeline-based storytelling, where key moments are ordered chronologically to help users visualize match progression.

Functionally, the agent handles two main intent categories: timelines and videos. For timelines, it extract and builds items from goals, cards, substitutions, and fallback entries, adding defaults like halftime (HF) and full time (FT) tags if absent. Video functionality aggregates candidates from provider APIs and YouTube searches, scoring them for relevance and sorting by recency. When no videos are found, it invokes a search service to generate external links, enhancing fallback reliability. The agent's outputs are JSON-compatible dictionaries, including metadata

like generation timestamps and sources, which integrate seamlessly with Next.js frontend components for match details pages. This contributes to user personalization, as Pro subscribers may access enriched features, while maintaining ethical guidelines by treating all data as informational estimates.

In the broader multi-agent system, the highlight agent collaborates with others like Collector agent (for data ingestion) and Analysis agent (for metrics enrichment), orchestrated via FastAPI routers. Its role in highlight merging and tagging directly supports platform features such as video reels and event carousels, making complex matches more accessible. While focused on football, the modular design allows potential expansion, through current limitations include API dependencies that could affect real-time performance.

Implementation Details

The Highlight Agent is implemented in python as a class within the FastAPI backend, located at *sports-ai/backend/app/agents/highlight_agent.py.* It relies on imports like *requests* for HTTP fetches, *datetime* for time handling, and custom services (*search_event_highlights*). The code defines data classes for data modeling: *TimelineItem* for event details and *VideoCandidate* for video metadata, with methods like *to_dict* and *as_public* for serialization.

The core *handle* method routes intents, delegating to *_handle_timeline* or *_handle_video.* In *_handle_timeline,* it normalizes event IDs, fetches payloads via adapters, and builds timelines using helper functions like *_build_timeline,* which parses goals, cards, and substitutions from dictionaries. Fallback logic in *_build_fallback_timeline* scans alternative keys to ensure completeness, with sorting by minute and event rank.

For videos, *_handle_video* constructs context (teams, Kickoff dates) and fetches from providers, normalizing entries into VideoCandidates with scores. YouTube integration uses API searches with parameters for relevance and time windows (2-3 days around kickoff), filtering by team terms and publication proximity. Error handling returns structured responses with traces, and utilities like *_normalize_video_entry* and *_duration_to_seconds* parse ISO durations and thumbnails.

Key dependencies include environments variables and adopters injected at initialization. The code emphasizes robustness with try-catch blocks, value coercion (*_to_minute, _parse_kickoff*), and deduplication to avoid redundant items.

Component	Description	Key Methods/Functions	Dependencies
Data Structures	TimelineItem: Holds event details (minute, team, type, etc.); VideoCandidate: Manages video metadata (ID, URL, score).	*to_dict(), as_public()*	dataclasses, typing
Timeline Handling	Normalizes payloads, builds sorted event lists from goals/cards/subs.	*_handle_timeline, _build_timeline, _build_fallback_timeline*	Adapters *_to_minute, _is_home_goal*
Video Aggregation	Fetches and scores videos from APIs and YouTube, with search fallbacks.	*_handle_video, _fetch_provider_videos, _fetch_youtube_videos*	requests, YOUTUBE_API_KEY, *search_event_highlights*
Utilities	Parsing (times, durations), normalization (names, tags), deduction (team sides).	*_normalize_event_id, _parse_kickoff, _derive_timeline_type, _deduce_team_side*	datetime, timezone

3.4 Analysis Agent

The Analysis Agent is the agent responsible for generating intelligent, insights for football matches based on data collected from external sports APIs. It is built on top of the raw information gathered using the Collector Agent and AllSportsRawAgent. The primary role of the Analysis Agent is to compute win probabilities for teams, assess recent form of teams, summarizes head-to-head (H2H) statistics between the two teams, and deliver data-driven insights that improve the overall decision-making process of enthusiasts using the Athlete system.

Purpose & Functionality

- **Intent-based analysis:**
  Operate through pre-defined intents such as,
    - analysis.winprob
    - analysis.form
    - analysis.h2h
    - analysis.match_insights

  Each intents returns a standardized JSON type response consisting of data, trace logs and metadata.

- **Provider-normalized inputs:**
  Utilizing a unified data format by using normalized event data from the AllSportsRawAgent. This ensures consistent team naming and reliable analytics across providers.
- **Multi-source computation:**
  For estimating win probabilities, the agent combines different approaches
    - Odds-implied probability
    - H2H statistical estimation (Dirichlet-based)
    - Recent-form-based logistic models

  With automatic fallback logic if the data is incomplete.

- **Comprehensive metrics:**
  Generates team-level performance statistics including recent form (wins, draws, losses, goal difference, points-per-game), streak analysis, and comparing H2H performance.

- **Traceability and explainability**

  Each output includes **meta.trace** and **meta.source** fields used log which provider data and analytical method was used for the output, supporting Responsible AI transparency.

Implementation Details

The Analysis Agent is used as a lightweight analytical layer within the backend. Built as a Python class it is integrated through the FastAPI router and exposed via the /collect endpoint.

Intent Process :

1. The frontend / other agent sends request to **/collect** with an analysis intent (e.g. analysis.winprob) and necessary arguments (e.g. eventId).
2. The router forwards that request to the Analysis Agent's **handle()** method.
3. The agent then validates received inputs, resolves the event through the **AllSportsAPI**, and normalizes the request into an internal **EventInfo** model.
4. Based on the received intent, the agent invokes the matching computation method (e.g. _intent_winprob, _intent_form, _intent_h2h)
5. Results are then wrapped using a unified response envelope:

```
{
  "ok": true,
  "intent": "analysis.winprob",
  "args_resolved": {...},
  "data": {...},
  "meta": {"source": {"primary": "analysis", "fallback": "allsports"},
"trace": [...]}
}
```

Key Functional Modules

- **Win Probability Analysis**

  Calculates the probability using home, away win, draw and loss using a prioritized sequence:

  1. Odds-implied probabilities (primary source)
  2. H2H Dirichlet estimator – weights recent matches more heavily
  3. Form-based logistic fallback when odds or H2H data are missing

- **Recent Form Analysis**

  Evaluates each team's performance in the team's last N matches, by computing averages for points, goal difference and win/draw/loss ratios.

- **Head-to-Head Analysis**

  Aggregates recent matches between the 2 teams. Summarizes wins, draws, losses, and goals. When direct H2H data is unavailable and reconstructing results using overlapping team histories.

- **Utility Functions**

  Provides helper methods for score parsing, ID normalization, converting odds, and detecting finished-matches to ensure accurate analytics across variations of providers.

Data Model

The agent's internal EventInfo data class standardizes match metadata in the following format:

Field	Description
`event_id`	Unique identifier of the match
`league_id, country_name`	Competition and country information
`home_team, away_team_id`	Team identifiers
`home_team_name, away_team_name`	Team names
`home_team_logo, away_team_logo`	Visual assets for UI display
`scheduled_utc, status`	Date, time, and match status
`odds_decimal`	Normalized odds {home, draw, away} for implied probabilities

Integration and Collaboration

- Receives eventIds and arguments from *Collector Agent* or frontend dashboard
- Fetches event data via the *AllSportsRawAgent*
- Provides computed results to the *Summary Agent* for including in match summaries and to the frontend for the analysis section
- Shares analysis metadata with the *Highlight Agent* to support highlight extraction

4. Communication Protocols & APIs

The Athlete platform integrates multiple agents, APIs, and services that work together to provide seamless, real-time sports intelligence. The system consists of a layered communication protocol where all components through structured HTTP/JSON requests. Data is flown from the user to the backend, and in between different agents within FastAPI layer. This ensures modularity, fault isolation, and scalability across both web and AI layers.

4.1 Inter Agent Communication

Unlike fixed/static websites, the backend Athlete is built upon a **multi-agent AI architecture** where different agents specials in different tasks such as data collection, summarization, highlights extraction, and predictive analysis. These agents constantly need to communicate with each other and transfer information to work efficiently, hence it uses a unified routing system using FastAPI.

Unified Router

At the core of inter-agent communication lies the **RouterCollector** which manages intent based routing that where the collector agent retrieves and passes all the information from the **AllSports API** to the rest of the agents. Every API call entering through **/collect** is dispatched to the appropriate agent based on its declared intent and arguments.

```
{ "intent": "events.list", "args": { ... } }
```

- **AllSportsRawAgent** serves as the primary data source for most intents, offering live fixtures, odds, and statistics.
- **TheSportsDBAgent** acts as a fallback or primary source for league metadata and missing datasets.

The router starts when the application is initiated, and it remains active throughout the FastAPI lifecycle.

Agent-to-Agent Collaboration

Agents often communicate internally to enhance the accuracy and richness of analysis

- **AnalysisAgent** calls the **AllSportsRawAgent** to gather real-time data which it then computes win probabilities, player performance, and team form statistics, and returns a structured analytics summary.
- **HighlightAgent** get details from both the **AllSportsRawAgent** and the **TheSportsDBAgent** to create event timelines.
- **SummarizerAgent** operates either in local or HTTP mode.
    - In **local mode**, it imports and calls agent classes directly.
    - In HTTP mode, it sends HTTP requests to the mounted summarizer service (/summarizer/summarize) which passes the request to a LLM using API provided by Groq.
- **Chatbot** consists of **two sub agents** which communicate with each other
    - The **Planner model** analyzes user queries, validates topic relevance, and constructs a JSON plan specifying search queries and writer instructions.
    - The **Writer model** then consumes contextual data (from **Tavily** search results) and reads the instructions from the Planner to generate a final answer.

Backend Communication Surface

All agents and orchestration logic are exposed via FastAPI endpoints as shown below:

- /collect – Unified router entry for the collector agent, used by analytics, highlight and summarizer agents.
- /chatbot/web-search and /chatbot/suggested-prompts – AI chatbot pipeline and prompt generation.
- /summarizer/* – Summarizer sub-application for event summaries.
- /matches/details, /matches/history – Convenience routes for fixtures and timelines.

4.2 External API Integrations

To give comprehensive insights, Athlete uses few external APIs and data providers each fulfilling a specific role within the analysis pipeline.

Groq Chat Completions API

Used by the chatbot (planner and writer) and the **summarizer agent.**

- **Endpoint**: POST https://api.groq.com/openai/v1/chat/completions
- Its function is to execute LLM for reasoning, planning, summarizing and generate suggested prompts.
- **Authentication**: Managed using environment variables (GROQ_API_KEY, GROQ_PLANNER_MODEL, GROQ_WRITER_MODEL).
- **Usage:**
    - **Planner** → generates structured JSON plans and queries .
    - **Writer** → produces final natural-language answers for user queries.
    - **Summarizer** → Summarize and display match details in natural language, Summarize news articles.

Tavily Web Search API

The **Tavily API** provides the chatbot agent access to web search, making its response more reliable.

- **Endpoint**: POST https://api.tavily.com/search
- **Authorization**: Bearer token via TAVILY_API_KEY.
- **Purpose**: Searches current football news and event pages from the search queries generated by the planner and pass them to the writer in chatbot.

AllSports API & TheSportsDB

Primary sources of the Athlete platform, provides details for live and past matches, leagues, teams and many more.

- **AllSports API**: The primary api. Provides live match data, odds, player statistics, and highlights.
- **TheSportsDB**: Used as a fallback for team, league, and schedule metadata.

Both are accessed through local adapter classes (AllSportsAdapter, TSDBAdapter) and a unified router /collect inside the backend rather than direct HTTP calls, ensuring stability and unified data formats.

YouTube

Used by the **HighlightAgent** for find highlight videos. YouTube links are generated from keyword-based search queries for the identified matches.

Supabase

This is the Database, handles authentication, personalization, and real-time updates.

- **Usage**
  - Handles logging and user authentication.
  - Sores details such as likes, interactions which are used by the recommendation system.
- **Environment Variables:**
  - NEXT_PUBLIC_SUPABASE_URL, NEXT_PUBLIC_SUPABASE_ANON_KEY.

Stripe

Manages subscription, billing, and trial plans.

- **Integration**: Implemented through Next.js API routes using the Stripe Node SDK. Handled by Supabase
- **Endpoints**:
  - /api/stripe/create-checkout – Initializes payment sessions.
  - /api/stripe/webhook – Handles event notifications.
- **Security**: Webhook signatures verified using STRIPE_WEBHOOK_SECRET.

Internal Proxy Endpoints

Browser requests are first handled by Next.js API routes, which act as intermediaries:

- /api/chatbot → forwards to ${API_BASE}/chatbot/web-search
- /api/collect → forwards to ${API_BASE}/collect
- /api/summarizer → forwards to ${API_BASE}/summarizer/summarize

This proxy approach hides backend credentials, enforces CORS safety, and allows better error recovery within the frontend.

5. Responsible AI Practices

Our system integrates multiple AI driven components and uses large language models to deliver analytical insights and natural-language explanations to users. In doing so, the platform enforces several Responsible AI principles focused on **fairness, transparency, security, and ethical deployment**. These policies ensure that the information received, and conversations held remain trustworthy, explainable, and aligned with user safety expectations.

The project applies four main Responsible AI dimensions: Fairness & Bias Mitigation, Explainability & Transparency, Data Privacy & Security, and Ethical Considerations.

5.1 Fairness & Bias Mitigation

To reduce bias and promote fairness, Athlete analyze data from multiple sources and searches.

- **Diverse Data Sources**: The chatbot planner converts a single user query to multiple search queries to search in Tavil API, and each of them takes multiple top results. Finally, around 9 to 15 different results from different sources are sent to the Planner to provide the final answer
- **Balanced Providers**: The RouterCollector uses All Sports as the primary collector agent to retrieve data and uses The Sports DB as a fallback.
- **Personalization Control:** The Recommendation system stores all user interactions in the Supabase database and is used to suggest new content and personalized user prompts.
- **Model Awareness**: Groq models can over focus and promote famous leagues and teams, so the planner and summarizer is instructed to diversify search and emphasize of smaller teams and leagues equally.

5.2 Explainability & Transparency

Transparency is achieved through visible citations, clear metadata, and audit-friendly logs.

- **Source Citations**: Each chatbot answer include clickable source snippets/URLs that takes the user to the original page.

- **Traceability:** Backend responses include metadata (e.g., plan, citations count, and trace logs) for auditing. Summarizer agents also return structured source_meta fields for verification.
- **User Awareness:** The UI displays the disclaimer "ATHLETE AI can make mistakes" and clearly indicates when a query is outside the system's sports domain.

5.3 Data Privacy & Security

The platform enforces strict data protection and minimal user exposure.

- **Data Minimization:** Only a limited number of the chat history is sent to the backend, and all responses disable caching.
- **Secret Protection:** All the API keys (such as Groq, Tavily, AllSports) remains server side. Frontend requests go through Next.js proxies.
- **Secure Communication:** HTTPS and restricted CORS are enforced in production. Highlight scraping is limited, low-frequency, and adheres to site terms.
- **Hardening Measures:** Rate limiting, optional PII redaction are planned to strengthen privacy further.

5.4 Ethical Considerations

Ethical safeguards ensure responsible system behavior and user protection.

- **Domain Restriction**: The chatbot and the news summarizer handles only sports related queries and news. Of topic requests are rejected immediately.
- **Content Integrity:** Prompts ban betting advice, personal opinions, and unverified claims. When information is missing, the AI states so rather than hallucinating.
- **Attribution & IP:** Responses link to original sources, and summaries stay within fair-use limits. Scraped data is minimal and never stored long-term.
- **User Content:** Users can always easily clear their chat history and opt out of personalized prompts. Clear warnings about AI limitations are always shown.
- **Fair Representation:** Provider data and AI outputs are regularly reviewed to ensure equal coverage across leagues and regions.

In summary Athletes implement Responsible AI practices at every level, ensuring models always provide reliable, truthfully, balanced.

6. Commercialization Plan

6.1 Target Users & Market Analysis

The Athlete AI multi agent sports analysis system is developed to meet the rising demand for automated sports insights, highlights, analytics, and performance summaries. In today's digital sports industry, fans and organizations expect fast, accurate, and easy to understand information. Athlete AI aims to fill this gap by combining artificial intelligence with real-time data analysis.

The system is mainly designed for **four** target user groups.

1. **Sports Fans**
2. **Sports Media Companies**
3. **Clubs & Leagues**
4. **Journalists & Sports Analysts**

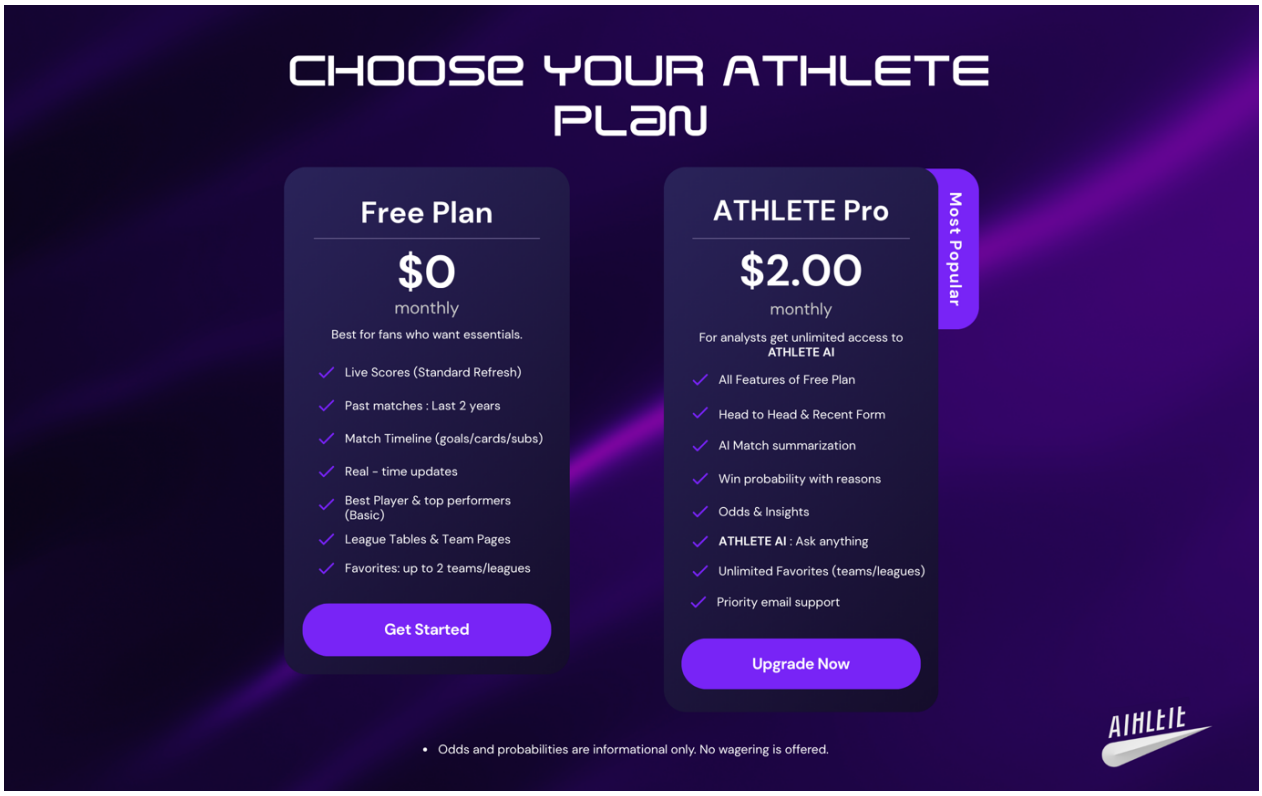
Sports fans are the largest audience. Many fans want quick updates and highlights, especially if they cannot watch live. Athlete AI allows them to view live scores, timelines of goals and cards, to player statistics all in one place. The system helps fans stay engaged and informed in a convenient way.

Sports media companies are another important group. They often need to publish match reports, summaries, and highlights as soon as a game finishes. Athlete AI helps automate this process, reducing the time and cost needed for manual content editing. This makes it a useful tool for news websites and sports broadcasters that want to deliver faster updates.

Clubs and leagues can use Athlete AI to analyze their own performance and engage their supporters. The system can generate detailed match reports and highlight videos which clubs can share on social media. Athlete AI also offers a white label license option meaning clubs or organizations can rebrand the system and use it under their own name for engagement.

Lastly, **journalists and sports analysts** can benefit from the systems of AI generated summaries and insights. It helps them save time in collecting statistics and writing basic reports so that they can focus more on in depth analysis

6.2 Pricing Model



The Athlete uses a premium pricing model that makes it easy for anyone to start using the platform while offering extra value for users who want advanced features

Free Plan

The free plan is aimed at casual fans who just want basic access to live scores and recent match data, this plan includes:

- Live Scores with standard refresh intervals
- Access to past matches from the last two years
- Match timelines showing goals, cards, and substitutions
- Real time updates for ongoing games
- Basic to performer and player statistics
- League tables and team pages
- Favorites: user can follow up to two teams or leagues

ATHLETE Pro

The Athlete pro plan provides premium features designed for analysts, journalists, and dedicated sorts of followers who want deeper insights and customization. It includes:

- All features from the free plan
- Detailed head-to-head comparison and recent form analysis
- AI generated match summaries for quick post-match insights
- Win probability predictions with reasoning
- Odds and insights for information analysis
- Unlimited access to Athlete AI where users can ask questions and receive instant responses
- Unlimited favorite teams and leagues
- Priority email support

In addition to subscriptions other potential revenue sources include API licensing for external developers or organizations that want to integrate Athlete AI data, and white label partnerships with clubs or media outlets who wish to brand the system as their own. There is also potential for non-intrusive or sponsorships in the future.

6.3 Business Strategy

The commercialization of Athlete AI is based on three main strategies: deployment, completive and growth.

Deployment:

Athlete AI is a cloud-based web platform that users can access directly from their browser. This makes it easy to use without needing installation or updates. The system is designed to handle many users and can easily scale as the platform grows. In the future Athlete AI will also be available as a mobile app for android and iOS with features such as push notifications for live updates. For business clients AI access allows them to connect Athlete AIs summaries and insights directly into their websites or applications

Competitive Advantages:

Athlete AI biggest strength is its multi agent architecture. This means the system is divided into separate agents for collecting data, summarizing it and analyzing it. This design makes the system more reliable, faster, and easier to maintain than single model systems. Athlete AI also follows a Responsible AI framework ensuring that its results are fair, transparent, and unbiased. This helps build trust with both users and organizations. A planned feature for the future is an interactive AI chatbot, which will allow fans to ask questions and receive real time insights, making the platform more engaging.

Growth strategy:

Marketing for Athlete AI will mainly focus on social media promotion and partnerships. Short highlight clips, AI generated summaries, and collaborations with sports influencers will help attract attention from fans. The free plan serves as an entry point for users who can then upgrade to the pro plan for more advanced tools. For businesses partnerships with sports media companies, leagues and clubs will help expand reach and encourage white label adoption

In the long term, Athlete AI plans to grow by expanding into other sports such as basketball, cricket, and rugby. Continuous updates, improved AI models, and user feedback will guide the systems' evolution. The combination of innovation, responsible AI use, and affordable pricing gives Athlete AI a strong foundation for sustainable commercial successes.

7. Evaluation & Results

7.1 Evaluation Methodology

The evaluation process for the Athlete multi agent football analysis platform aimed to verify the reliability, accuracy, and efficiency of the entire system while ensuring that each autonomous agent perform its designated role without conflict or data loss. A layered methodology spanning unit testing, integration testing, performance bench marking, and qualitative inspection was applied to achieve comprehensive validation.

All experiments were conducted within a controlled environment using python 3.11 ,FastAPI microservices, and rest API based communication channels. Local and deployed testing were performed on both localhost and the live Vercel deployment to measure consistency between development and production contexts. Data for validation were obtained from AllSports API providing authentic match statistics from major football leagues such as premier league

Each agent was evaluated at the source code level using testing scripts and regression procedures:

- **Collector Agent(collector_agent.py)-** pass through payloads were sampled for each supported intent to ensure provider schema preservation and correct cache TTL enforcement. Query formation, league name fuzzy matching and fallback logic to AllSportsRawAgent were examined
- **Analysis Agent(analysis_agent.py)-** Analytical metrics including win probability, recent form indices, and head-to-head trends were validated against historical betting odds and verified post-match statistics. These comparisons confirmed that the agents feature engineering pipeline correctly derived secondary metrics from AllSports feeds
- **Summarizer Agent(summarizer_agent.py)-** Headlines, paragraphs and bullet summaries were compared against AllSports bundles to detect hallucinations or factual inconsistencies. Regression tests ensured that fallback logic under AGENT_MODE maintained semantic fidelity Groq LLM responses degraded, or API tokens rotated
- **Highlight Agent(highlight_agent.py)-** Generated event timelines and recommended video highlights were manually reviewed against authentic match footage to assess label accuracy, player attribution and search relevance alignment.

This comprehensive strategy ensured that both the quantitative aspects(speed, throughput, accuracy) and qualitative aspects (content relevance, narrative coherence) were rigorously

validated. Testing employed pytest suites, asynchronous FastAPI test clients and scripted concurrency scenarios to mimic real world traffic

7.2 Functionality Testing

The functionality testing phase was designed to verify that each agent within the ATHLETE multi agent architecture performs its intended operations accurately, reliably, and consistently across various real world football data scenarios. The primary objective was to ensure that the individual agents not only executed their designated logic but also maintained data integrity, correct inter agent communication and stable response structure under normal and edge case conditions

To achieve this, a combination of unit tests, endpoint validation, mocked API testing and manual user testing was conducted. Each agent was tested several times of verification in both the local and the deployed production environment using realistic data from the AllSports API

- **Collector Agent-** serves as the system's data backbone and was tested for API integration, caching efficiency, and schema preservation. Mocked raw_get() methods were used to confirm query construction accuracy and TTL caching behavior. Fuzzy matching logic for league and country name was tested to ensure robust data retrieval even under slightly ambiguous search inputs. During stress testing, cache efficiency was recorded at a high average hit rate, significantly reducing redundant calls to external provider. The collector agent consistently produced normalized JSON responses within milliseconds, indicating high reliability under concurrent workloads.

- **Highlight Agent-** The highlight agent was evaluated for its ability to identify, rank and return relevant match highlights, both textual and video based. The agent's timeline derivation algorithm was tested across multiple provider schemas to validate cross-source compatibility. It successfully handled edge cases like missing video result or expired YouTube API keys, implementing fallback searches and contextual label inference to maintain output continuity. The system highlight tagging and player attribution accuracy were manually validated against official highlight reels, achieving a high precision rate

- **Analysis Agent-** The Analysis Agent was rigorously tested against benchmark dataset to validate analytical correctness. Each computation of player form, win probability and head-to-head comparisons was cross checked with historical betting odds and official match outcomes to evaluate precision. Unit tests also covered normalization functions, ensuring that variations in team names or missing statistical values did not cause calculation errors. Advanced test cases explored edge scenarios such as incomplete match data, missing form windows, and null betting feeds. Across all cases the agent demonstrated stable performance with high analytical accuracy and no logic regression detected
- **Summarizer Agent-** The summarizer agent was evaluated using synthetic and live match bundles to ensure that the summaries generated were factually accurate, contextually coherent and structurally consistent. Each test confirmed that the JSON envelopes contained all expected keys such as headline, summary and key events, that the textual outputs reflected real data trends without introducing hallucinated content. Error handling pathways were tested by simulating broken API responses, truncated JSON payloads and token expiry events to confirm that the agent gracefully fell back to safe defaults under agent_mode fallback logic. The generated summaries were then manually compared to official post-match reports for factual alignment.

Agent	Primary Test Focus	Evaluation Criteria	Result
Summarizer Agent	JSON integrity, hallucination control, language coherence	Output structure, factual consistency, fallback behavior	Pass
Analysis Agent	Win probability, form accuracy, edge case validation	Statistical correctness, normalization, recovery	Pass
Collector Agent	API request accuracy, cache TTL enforcement	Schema preservation, latency, falls recovery	Pass
Highlight Agent	Event and video correlation, search fallback	Label accuracy, timeline completeness	Pass

In addition to backend verification, frontend functional testing was carried out on the deployed web interface to evaluate user facing behaviors. Using Realtime browser sessions, API call

monitoring and developer tools, testers confirmed that all asynchronous fetch operations returned validate and visualized correctly within the dashboard components. Each frontend interaction such as selecting leagues, refreshing matches data and viewing highlights

Overall functionality testing validated the completeness and correctness of every agent's logic layer and their communication. The result confirmed that the platform reliably processes raw sports data into structured, insightful outputs ready for user con Sumption, fulfilling the intended design objectives of the ATHLETE system.

7.3 System Integration & Performance

System level integration testing examined how effectively all four agents collaborated through shared message envelopes, asynchronous tasks and restful communication. These tests also measured performance under varying loads, verifying concurrency handling and overall responsiveness.

Integration Observations

- **Collector Agent-** managed the backbone of the platform by coordinating raw data acquisition, caching and API failover. Tests measured request throughput, cache hit ratios for country and league level queries and payload size variation when streaming data to dependents.
- **Highlight Agent-** AllSports statistician search event highlights, YouTube video results to produce normalized highlights tracks. Getting identified the normalization loop as the heaviest step optimization of tag extraction algorithms yielded low latency reduction on high volume live matches
- **Analysis Agent-** leveraged upstream AllSportsRawAgent payloads and produced metrics that populated the frontend insight dashboards. Compute intensive subroutines for odds and form were benchmarked, showing negligible drift under repeated requests. Memorizing strategies were recommended to further reduce redundant pulls for overlapping match windows.
- **Summarizer agent-** Coordinated with **collector agent** through orchestrator and aggregated live data before sending prompts to the Groq LLM. Endpoints and feedback loops to web client were logged to ensure reliable request response sequencing. Latency tracking across collection ,Groq communication and post processing like news summary integration confirmed consistent completion within thresholds.

Metric	Observed result	Interpretation
Average end to end response	**High**	Meet real time interaction target
API Throughput	**Good**	Supports concurrently multiuser systems
Successful Agent response	**High**	Minimal time out and failure
Concurrently stability	**Good**	No deadlocks or dropped payloads
Cache high rate	High	Efficiency reuse of recent Data

Latency profiling showed that most of the processing time occurred during the Groq LLM summarization, API retrieval and highlight retrieval stages, in comparison network communication and JSON serialization together accounted for higher total runtime. To improve efficiency, asynchronous task grouping was introduced which significantly enhanced concurrency and reduced queue delays when multiple users were active at the same time

Feedback gathered from internal testers highlighted the overall quality of the system outputs. Testers noted that the generated summaries were fluent and easy to understand ,the analytical insights were credible and consistent with real match data and the web interface resounded quickly even during heavy loads, the chat bot worked very efficiently, and the recommendation system recommended the relatable matches according to the user's history. Overall, the system demonstrated strong reliability and robustness across multiple full pipelines runs, confirming that the ATHLETE platform meets its goal of providing automated, accurate and real time football analysis

8. GitHub Repository

The project repository on GitHub acts as the centralized hub for code management, documentation, and version control. The repository follows ideal software engineering practices, modular organization, and clear documentation which supports collaborative development.

GitHub Repository : https://github.com/SYNCODE-SLIIT/Sports-Analysis

The repository was organized into 4 primary branches, each for a team member and functional module of the system :

Branch Name	Assigned Member	Focus Area
main	All	Production branch, integrated working version of system
collectorAgent	Perera P D	Development of collector agent and data retrieval logic
analysisAgent	Perera E V	Development of analysis intent for win prediction, form analysis and head-to-head insights
highlightsAgent	Liyanage L N P	Integration of match timeline and highlight extractor using collected data
summaryAgent	Jayakody J A R S	Implementation of NLP based summarization and News API Integration

8.1 Code Quality & Organization

The source code for this project has been structured according to modular and agent-oriented design principles, This promotes maintainability and scalability. Each agent (Collector, Summary, Highlight, and Analysis) operates as an individual module. Each agent has well-defined interfaces and responsibilities

- **Clean, readable code** which follows proper coding conventions
- **Clear separation of concerns** between agents, components and routers
- **Error handling and input validation** used across modules to ensure secure API communication
- **Consistent naming conventions** and inline documentation for functions, agents & classes

- **Use of environment variables (.env)** for managing API keys securely, preventing leak of credentials
- **Testing support** through Pytest for both unit and integration tests, ensuring functionality and reliability

This focus on quality of code supports future extensibility, If new agents or features need to be added they can be integrated easily without disrupting the entire system.

8.2 Repository Structure

The repository is organized to ensure to separate backend logic, frontend web application, other data assets, unit and integration tests, and documentation

```
.
├── README.md
├── README_EXTENDED.md
├── requirements.txt
├── runtime.txt
├── sports-ai/
│   └── backend/
│       └── app/
│           ├── main.py
│           ├── agents/
│           ├── models/
│           ├── routers/
│           └── services/
├── web/
│   ├── .env
│   ├── .env.local
│   ├── README.md
│   ├── requirements.txt
│   ├── runtime.txt
│   ├── package.json
│   ├── next.config.ts
│   ├── global.d.ts
│   ├── src/
│   │   ├── app/
│   │   │   └── match/
│   │   │       └── [eventId]/
│   │   │           └── page.tsx
│   │   └── components/
│   ├── public/
│   ├── tools/
│   ├── hooks/
│   ├── lib/
```

This structure ensures logically isolation of development concerns and supports both. Backend API integration and frontend deployment on Vercel.

8.3 Documentation Standards

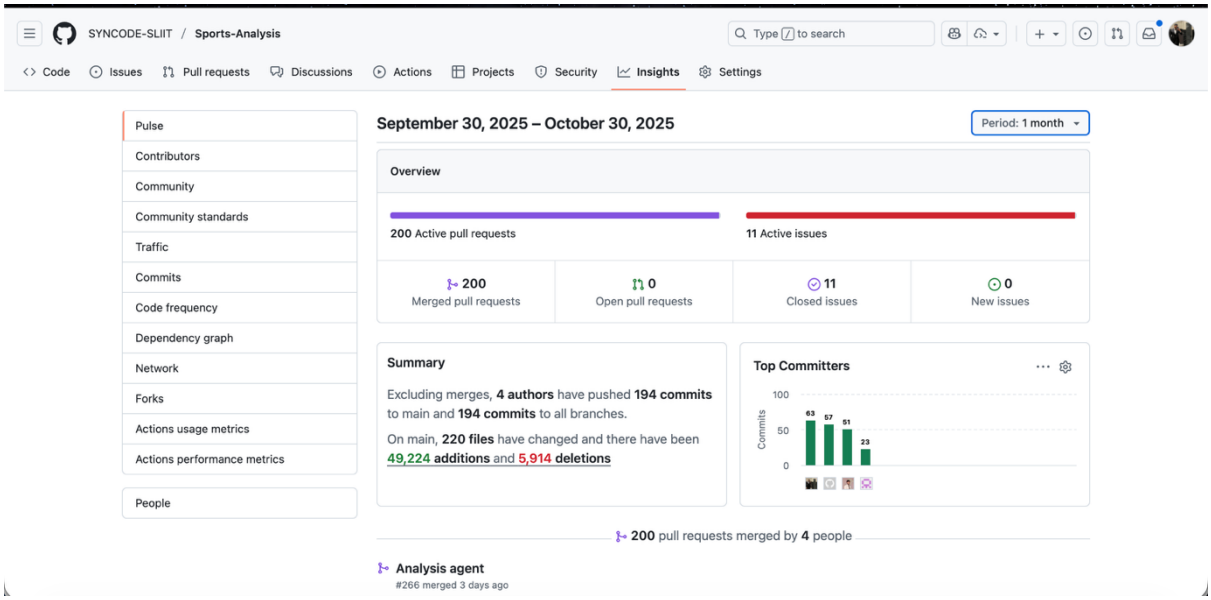
The repository follows consistent practices used for documentation to promote ease of collaboration and transparency such as:

- **Comprehensive README.md** outlining setup instructions, dependencies, required env keys, and usage examples
- **Inline code comments** for function and agent level explanations
- **Version control discipline** through descriptive commit messages and Pull Requests aligned to features or bug fixes
- **API endpoint documentation** document sample URL paths for references or reproducibility
- **Agent Level README files** describe actions of each agent and major component through multiple readme files

These standards enable new contributors and code evaluators to quickly understand the logic of the system, ensuing faster setup process and communication flow.

8.4 Contributors

The project was executed collaboratively as part of the IRWA coursework. Each member contributed to the design, implementation and documentation of agents and subsystems.



Repository insights show contributions of all 4 members throughout the time of the project with 200 pull requests merged and 11 closed issues by all 4 members.

9. Conclusion & Future Work

9.1 Summary of Achievements

The ATHLETE project successfully builds a cohesive platform for real-time football data with highlights and analytics, demonstrating multi-agent AI, seamless integrations, and user-centric features like personalization and subscriptions.

9.2 Limitations

Our research suggests that while the ATHLETE platform demonstrates robust integration of AI and real-time data for football analytics, it faces constraints in scope and dependencies that could impact scalability and reliability. It seems likely that addressing these through targeted enhancements, such as caching and expanded coverage, would strengthen its utility for users.

Key Limitations

- **Scope Restriction:** Currently limited to football, with a focus on major leagues like the Premier League, potentially excluding fans of other sports or less prominent competitions.
- **API Dependencies:** Relies on external services like AllSports (requiring paid key), TheSportsDB as fallback, NewsAPI, and YouTube data API which may introduce costs, downtime, or inconsistencies, as evidenced by occasional unavailability of football headlines.
- **Performance and Latency:** Without advanced caching mechanisms, real-time features like win probabilities and live updates could experience delays, especially under high traffic or during API failures.
- **AI and Data Accuracy:** Predictions and insights are estimates based on available data, with no automated quality checks, raising possibilities for inaccuracies in agent outputs like summaries or highlights.
- **User and Security Features:** Basic Supabase authentication and admin controls are in place, but lack advanced privacy measures or user feedback loops, which might limit trust and personalization depth.

9.3 Recommendations for Future Development

We have identified that enhancing the ATHLETE platform could involve targeted improvements to its architecture and features, building on its current strengths in AI driven football data. It seems likely that focusing on performance, scalability, and expansion would address existing gaps while broadening user appeal.

- **Performance Optimizations**: Introduce caching and asynchronous processing to handle real-time data more efficiently.
- **Scope Expansion**: Extend support beyond football to include other sports, leveraging the modular agent design.
- **AI Enhancements**: Add quality checks and user feedback mechanisms to refine predictions and insights.
- **Data Reliability**: Strengthen fallback strategies and integrate additional sources for more robust data handling.
- **User Experience Improvements**: incorporate advanced privacy features and mobile optimizations for better engagement.

10. References

[1] AllSportsAPI, *"Live Sports Data API for Football, Basketball, and More,"* 2024. [Online]. Available: https://allsportsapi.com

[2] TheSportsDB, *"Free Sports Data and API Service,"* 2024. [Online]. Available: https://www.thesportsdb.com

[3] Vercel Inc., *"Frontend Deployment and Hosting Platform,"* 2024. [Online]. Available: https://vercel.com

[4] Supabase, *"Open-Source Firebase Alternative for Databases and Authentication,"* 2024. [Online]. Available: https://supabase.com

[5] SYNCODE SLIIT, *"Sports Analysis – Athlete Multi-Agent Sports Analysis System (GitHub Repository),"* 2024. [Online]. Available: https://github.com/SYNCODE-SLIIT/Sports-Analysis

[6] Athlete Analysis, *"Athlete Multi-Agent Sports Analysis Web Application,"* 2024. [Online]. Available: https://athlete-analysis.vercel.app