

Software Requirements Specification

for

StatSync

By: Darragh Gorman, Hugh Quigley

26/11/2025

Table of Contents

Introduction.....	1
1.1 Purpose.....	1
1.2 Document Conventions.....	1
1.3 Intended Audience and Reading Suggestions.....	1
1.4 Product Scope.....	1
1.5 References.....	1
Overall Description.....	2
2.1 Product Perspective.....	2
2.2 Product Functions.....	7
2.3 User Classes and Characteristics.....	8
2.4 Operating Environment.....	8
2.5 Design and Implementation Constraints.....	10
2.6 User Documentation.....	11
2.7 Assumptions and Dependencies.....	11
External Interface Requirements.....	12
3.1 User Interfaces.....	12
3.2 Hardware Interfaces.....	13
3.3 Software Interfaces.....	14
3.4 Communications Interfaces.....	14
System Features.....	15
4.1 Player Data Import (CSV Upload).....	15
4.2 Real-Time Team Communication.....	16
4.3 Real-Time Match Event Logging.....	16
4.4 Automated Performance Feedback (Analytics Engine).....	17
4.5 Video Timestamp Jumping & Clip Extraction.....	18
4.6 User Onboarding & Interactive Tutorial.....	18
4.7 Cockburn Use Cases.....	19
Other Nonfunctional Requirements.....	22
5.1 Performance Requirements.....	22
5.2 Safety Requirements.....	23
5.3 Security Requirements.....	23
5.4 Software Quality Attributes.....	24
5.5 Business Rules.....	24
Timeline.....	25
Other Requirements.....	26
Appendix A: Glossary.....	26
Appendix B: Analysis Models.....	27
Appendix C: To Be Determined List.....	30

1. Introduction

1.1 Purpose

The StatSync Football Analytics App enables football analysts to log match events (possession won/lost, shots, duels, key passes, errors, etc.) for individual players, pitch zones and the team during live matches using a device such as an iPad interface. Each action is timestamped with the match timer, allowing managers to view team and player statistics in real time via a separate iPad dashboard, and to access precise event moments after the match for fast, efficient review. This app focuses on the match analytics functions—namely the analyst and manager modules—excluding broader club management tools and integrations.

1.2 Document Conventions

To guarantee readability and clarity throughout the document, requirement identifiers are bolded. To emphasise their introduction into the specification, key system terms like event, timestamp, and dashboard are italicised when they first appear.

According to standard requirements-engineering terminology, recommended or optional behaviours are expressed using **SHOULD**, while mandatory system behaviours are indicated using the term **MUST**. To ensure consistency, timestamps synchronised with the official game clock are used to record all match-related events.

When it comes to data visualisation requirements, the intended graphical format—such as pie charts or bar charts—is specified in parentheses right after the requirement text.

Sub-requirements inherit the priority given to their corresponding high-level parent requirement, unless otherwise specified.

1.3 Intended Audience and Reading Suggestions

This Software Requirements Specification (SRS) is intended for software developers, system engineers, UI/UX designers, quality assurance testers, football analysts and team managers involved in the StatSync platform.

Section 2 provides an overall description of the system, while Sections 3 and 4 give detailed requirements for the analyst and manager modules, respectively. Subsequent sections outline the system architecture and the specifications for each major feature.

Technical stakeholders, including developers and testers, should focus primarily on the architectural descriptions, functional requirements, and interface specifications. Football analysts and managerial staff are directed towards the workflow descriptions, usability considerations, and analytics-related features.

New readers are advised to begin with this Introduction before continuing to the system overview and the sections most relevant to their role.

1.4 Product Scope

StatSync digitizes the process of football match event recording, enabling analysts to log granular match actions by player, event type, pitch section, and real-time match timer, then instantly share the results with managers for tactical decision-making. Key benefits include rapid real-time analytics, easier review of key match moments using timestamps, enhanced collaboration between analysts and managers, and additional features including player comparisons, video clipping, automated reports, and trend analysis of historical data. These align with modern football club goals of improving match insights, speeding communication, and leveraging technology for competitive advantage.

1.5 References

IEEE Computer Society. *IEEE Std 830-1998 – Recommended Practice for Software Requirements Specifications*. IEEE Press, 1998.

ISO/IEC/IEEE (2024). *Systems and Software Engineering — Life Cycle Processes — Requirements Engineering*. Available at: <https://www.iso.org/standard/72047.html>

Pressman, Roger S., and Maxim, Bruce R. *Software Engineering: A Practitioner's Approach*, 8th Edition. McGraw-Hill, 2014.

React Native Documentation. *React Native: Components, APIs and Guides*.
<https://reactnative.dev/>

Django Software Foundation. *Django 4.x Documentation*.
<https://docs.djangoproject.com/>

Pappalardo, Luca et al. “A Public Data Set of Football Match Events.” *Scientific Data* 6, Article 236 (2019).

Credible academic source for sports-event data modelling.

StatsBomb. *Event Data Specification (Open Data Release).*

<https://statsbomb.com/>

Industry-standard sports analytics event model.

StackOverflow / StackExchange (2024). *Requirements Engineering & System Design Q&A.*

Available at: <https://softwareengineering.stackexchange.com/>

2. Overall Description

This section provides an overview of how StatSync fits into its intended environment, including its purpose, main functions, user groups, operating conditions, constraints, documentation, and project assumptions.

2.1 Product Perspective

StatSync is a standalone application designed to support real-time football match analysis by connecting analysts and coaches through a shared, synchronised system. It uses a client–server model where the front end is built in React Native and the back end runs on Django with WebSocket support for live communication. All match events, user actions, and analytics are stored in a cloud database so they can be accessed during and after matches. The system brings together event logging, video review, messaging, and analytics into a single platform, replacing traditional manual note-taking or separate tools that clubs often use. While StatSync operates independently, it is built to work alongside existing match video sources and club workflows by allowing video uploads and data export when needed. The application provides several key features that work together across different subsystems to support real-time analysis, event logging, communication, and performance review. These subsystems are described below.

Analyst Module

The Analyst Module allows analysts to record match events with accurate timestamps and pitch-zone selections, ensuring reliable and consistent data capture throughout the match.

Analysts can correct or update previously logged events if mistakes are made, helping to maintain clean and accurate match records. The module also supports full offline logging, so analysts can continue recording events even if the device temporarily loses connection, with all data automatically synchronised once connectivity returns. In addition, analysts can upload match videos and link them to specific match IDs, allowing detailed post-match review and smooth navigation to key timestamps.

Manager Module

The Manager Module provides coaching staff with real-time dashboards showing overall team performance and individual player metrics. These dashboards are updated instantly as events are logged, giving managers a clear view of how the match is progressing. An interactive timeline with colour-coded event markers allows users to revisit important match moments quickly, and the video panel supports jumping directly to timestamps linked to recorded events. The module also generates automatic tactical recommendations through the analytics engine, supporting informed decision-making during and after the match. Managers can also compare performance across previous matches to identify patterns, improvements, and areas that may need attention.

Real-Time Communication Module

The Real-Time Communication Module enables instant messaging between analysts and managers using WebSocket technology, ensuring communication happens without delay. This module also keeps all connected devices synchronised, so any new events, statistics, or updates appear immediately for everyone involved. This helps both sides maintain a clear shared understanding of the match, allowing for quick tactical discussions and coordinated decision-making.

Analytics Engine

The Analytics Engine processes match events and statistical data to generate meaningful insights, such as performance metrics, trends, and significant patterns that emerge during play. It presents this information through visual outputs including heatmaps, charts, and player summary panels, making the data easier to understand. The analytics update automatically as new match events are recorded, ensuring that all displayed insights remain accurate and relevant throughout the match.

Video Review Module

The Video Review Module allows users to upload and process full match videos and link them directly to the recorded event timestamps. This makes post-match review much faster, as users can jump straight to key moments by selecting events on the timeline. The module is also designed with future features in mind, such as automatic clip extraction and highlight generation, making it flexible for future development.

User & Access Control

The User and Access Control subsystem manages authentication and session validation to ensure secure access to the platform. It assigns role-based permissions, meaning analysts, managers, and coaches are only able to access the parts of the system that match their responsibilities. This prevents accidental modification of data and ensures the system remains secure and consistent.

2.2 Product Functions

The primary goal of this application is to provide an intuitive dashboard for users, based on real-time analytical data, which is to be stored securely in a cloud database for future analysis. The system is designed to support multi-platform functionality and enables communication between a React Native front-end and a Django back-end by utilising web sockets and servers to allow real-time updates and synchronization.

The application will also include an advisor section that interprets analytical data provided by analysts and offers data-driven recommendations to help managers make informed decisions and improve overall performance.

To further emphasize the need for real-time analysis and decision making, the app will incorporate a real-time chat interface between managers and analysts, allowing direct communication about reports, insights, and performance actions all within the same platform.

The main elements covered by the application will be as follows:

- Real-time analytics and visualization through dynamic, interactive dashboards.
- Cross-platform compatibility, ensuring access across web and mobile devices.
- WebSocket communication for instant updates between the front-end and back-end.
- Cloud database integration for secure storage and historical data analysis.
- Advisor module for generating performance insights and actionable recommendations.
- Chat and messaging system enabling real-time communication between managers and analysts.

2.3 User Classes and Characteristics

We project that the main users of this application will be sports managers, coaches, and performance analysts who require a real-time analytics platform to track, evaluate, and improve athlete and team performance.

These users will need a tool that allows them to visualize data, communicate insights, and make evidence-based decisions without needing in-depth technical knowledge of programming or data engineering. The application therefore aims to provide an intuitive and accessible interface that simplifies complex analytical processes, allowing users to focus on tactical insights and performance outcomes rather than technical details.

The key user classes are as follows:

- **Sports Managers / Team Managers:**
These users would mainly be tasked with reviewing team performance as well as strategic decision-making. They will primarily use the dashboard to review team and player statistics, compare match data, and evaluate progress over time. They are expected to have limited technical expertise but will access the system frequently to support match planning and resource allocation.
- **Coaches:**
Coaches will use the platform daily during training and competitive fixtures. They require clear, real-time visualizations that support tactical analysis and player performance monitoring. Their use will focus on fast interpretation and immediate feedback.
- **Performance Analysts:**
Analysts will have some form of technical use in the application. They will collect, process, and upload data, to then be further analysed. Analysts will have more system privileges as they are responsible for maintaining the accuracy and quality of the underlying data.

2.4 Operating Environment

The application is designed to be operational across multiple platforms, supporting both web access and mobile deployment. The architecture follows a client-server TCP model, in which a React Native front end communicates with a Django back end through WebSockets. Real-time message passing is handled by a Redis server, which manages connection and supports live user interactions and analytics updates.

All development work is carried out within Visual Studio Code (VS Code), where multiple integrated terminals are used to run the front end, back end, and Redis server concurrently. This setup mirrors the behaviour of the full production environment and allows seamless

debugging, testing, and coordination between the client and server layers during implementation.

From a hardware perspective, the system is intended to run on standard desktop or laptop computers during development, while end-users can access the application on mobile devices such as smartphones and tablets. The back end and database components are deployed on a cloud-based infrastructure to ensure high availability, scalability, and stable performance for multiple simultaneous users.

The system supports a wide range of operating systems. On the client side, users may run the application from any modern web browser on Windows 10/11, macOS, or Linux machines. For mobile access, the front end is delivered through Expo, enabling installation and demonstration on Android devices running version 9 or above and iOS devices running version 13 or later. On the server side, the Django back end and Redis services operate within a Linux environment, typically Ubuntu 22.04 LTS or an equivalent development setup using Python 3.11, providing a consistent and lightweight execution environment.

The software stack includes React Native (via Expo) for the front-end interface, employing JavaScript/TypeScript, HTML5 and CSS3 to produce dynamic and responsive screens. The back end is implemented using Django, with Django Channels providing native support for WebSocket communication. Redis functions as the message broker, ensuring efficient and ordered data transfer between connected clients. MongoDB Atlas, hosted on an AWS cloud cluster, serves as the primary database for persistent storage and retrieval of analytics data and user information. Development, testing, and debugging workflows are centred entirely within VS Code.

Because the system relies on real-time data synchronisation, a stable internet connection is required throughout use. Communication between the client, server, Redis broker, and cloud-based database is carried out over encrypted HTTPS and WSS protocols to guarantee secure transmission and protect user data.

2.5 Design and Implementation Constraints

The design and implementation of the system are influenced by a number of constraints that define the functionality of the project. The overall software stack was chosen in the early planning stages, meaning that the front end must remain implemented in React Native while the back end must continue to operate using Django and WebSocket-based communication through Redis. All components must therefore integrate cleanly with this stack and must

respect the requirement for continuous, real-time bidirectional data flow between clients and the server.

MongoDB Atlas, hosted on an AWS cloud cluster, is the designated database platform for the application. This imposes a non-relational, document-oriented data model on all stored information. Any additional data structures introduced during development must therefore follow a document-based design rather than traditional relational tables, ensuring consistency with the underlying database format and the constraints of the cloud-based deployment.

All development occurs within Visual Studio Code using its multi-terminal environment to run the front end, back end, and Redis services simultaneously. This configuration standardises the development process but also requires that all developers maintain consistent environment variables, dependency versions, and system configurations to avoid incompatibilities between setups.

The deployment environment introduces further constraints. The mobile front end is delivered through Expo, which limits access to certain native device modules and requires an active internet connection during live update sessions. The Django back end and Redis server must operate on cloud-compatible Linux systems capable of maintaining stable WebSocket connections for the duration of user activity.

Because the application handles real-time analytics and continuous data streaming, overall performance is influenced directly by server bandwidth and the user's network connection. Devices with reduced processing power, such as older mobile phones, may experience decreased responsiveness during high-intensity operations such as data visualisation or rapid update cycles.

Security requirements impose additional restrictions. Sensitive performance data must be encrypted in transit using HTTPS and WSS, and authentication must be carried out using Django Forest Framework and JWT tokens. No identifiable or confidential information may be stored or transmitted without adequate protection, and the system must operate in alignment with GDPR and general data-handling best practices.

Finally, the system's long-term maintainability depends on strict adherence to established naming conventions, modular programming practices, and clear separation between front-end and back-end logic. Because the expected maintainers of the system will continue to work with React Native, Django, and MongoDB, the use of alternative frameworks is not permitted within the scope of this project.

2.6 User Documentation

The application will provide a built-in Help section to aid first-time users. This will include a short, step-by-step walkthrough that appears on the user's first visit, explaining the main

features and how to navigate the interface. Additional lightweight documentation will also be provided. All documentation will be delivered in digital format as part of the application.

2.7 Assumptions and Dependencies

The project assumes that all required third-party services and tools such as the WebSocket server, cloud database, and deployment environment are available, supported, and compatible with the current technological stack. It is also assumed that development tools remain stable across all systems used during the project.

The project depends on external services such as the cloud database provider, authentication services, and any libraries or frameworks integrated into the system. Changes to, or failures in, these external components may impact system functionality or require adjustments to the implementation.

3. External Interface Requirements

3.1 User Interfaces

The system includes several main user interfaces that support the different tasks carried out by analysts and coaching staff during a match. All screens are designed to be clear, easy to use, and efficient during fast-paced match situations.

A. Analyst Interface

The Analyst Interface shows a full-pitch grid divided into six selectable zones so analysts can quickly link each event to the correct area of the pitch. It also includes an event panel with easy-to-recognise icons for common actions such as passes, shots, errors, and duels. A match timer and half indicator are always visible so events can be logged with accurate timestamps without leaving the main screen.

The interface also displays an offline indicator and a sync badge, letting analysts know when the app has lost connection and when events have been successfully uploaded again. Undo and retry pop-ups help correct mistakes without interrupting the flow of logging events.

B. Manager Dashboard

The Manager Dashboard provides an overview of team performance through clear summary panels showing key match statistics. It includes player-specific sections that present individual performance data, trends, and important contributions.

A timeline with colour-coded markers allows coaches to review important match moments quickly, and the video playback area supports jumping directly to timestamps linked to specific logged events. A real-time chat panel is also available on the right side of the screen so coaches and analysts can communicate instantly during the match.

C. Global UI Requirements

All screens must use a consistent colour scheme, layout structure, and visual hierarchy so the app feels unified and intuitive. Actions that could cause data loss or other important changes must display a confirmation window before proceeding. Error messages should be shown using toast notifications or small inline messages so users are informed without interrupting their workflow.

3.2 Hardware Interfaces

The application works on a wide range of mobile devices because React Native supports responsive layouts. Even though it runs on many screen sizes, the interface is designed to work best on medium-sized tablets such as iPads running iOS 15 or above, and Android tablets running Android 10 or higher. These devices provide enough screen space for selecting pitch zones, logging events, and viewing analytics comfortably. Devices should have at least 4 GB of RAM so the system can run smoothly during real-time updates and video playback.

The system also works on smaller smartphones, but some actions, like tapping pitch zones or using detailed dashboards, may be less convenient because of the smaller screen size. The layout adjusts, but a tablet offers the intended experience. A stable Wi-Fi or mobile data connection is required so that real-time event logging, analytics, and syncing work without interruption. If the connection drops, the app continues running offline and will sync everything once it reconnects.

For video features, the system assumes match footage comes from external storage such as USB devices or cloud services. The application does not use or depend on the device's camera hardware.

3.3 Software Interfaces

The system uses several external software components that support data storage, communication, and the front-end interface. The backend uses MongoDB Atlas with PyMongo drivers to store and retrieve match information. If these drivers change or stop being supported, updates may be needed to keep the system working correctly.

Real-time communication is handled through WebSocket connections, using Django Channels and Redis. These allow fast message updates and live event syncing. If WebSocket behaviour or Redis support changes, parts of the communication system may need to be updated.

The front-end is built using React Native along with libraries for charts, animations, and state management. These libraries are fixed to stable versions for reliability. If any library releases a major update, it may require testing or changes to keep the interface and communication features working as expected.

3.4 Communications Interfaces

The application requires a stable internet connection so that real-time analytics, event logging, and messaging all work correctly. Communication between users' devices and the server uses WebSocket connections for live updates and HTTPS/WSS for secure data transfer.

Low-latency networks are important so that events, chat messages, and analytics appear instantly on all devices. If there is high latency, packet loss, or blocked ports, real-time syncing may slow down or temporarily stop. To reduce these issues, the application automatically tries to reconnect and stores any unsent events until the network connection returns.

All data sent between the client and server is encrypted to keep match information secure. The system also relies on proper DNS and certificate validity to keep secure connections active.

4. System Features

This section describes the core functional capabilities of the StatSync application. The main features focus on supporting real-time match analysis and smooth communication between analysts and coaching staff.

These include uploading and validating player data through CSV files, providing a live chat system using WebSockets, and allowing analysts to record and synchronise match events across connected devices.

The system also includes an analytics engine that generates performance insights and recommendations based on logged events.

Video support enables users to jump directly to key timestamps linked to match actions, and a first-time user tutorial guides new users through the main features of the app, with the option to replay the walkthrough at any time.

4.1 Player Data Import (CSV Upload)

4.1.1 Description and Priority

Allows managers to upload CSV files containing player details. The system validates and stores the data for analytics.

Priority: High

4.1.2 Stimulus/Response Sequences

- User uploads CSV file: System validates structure and fields.
- CSV is valid: System imports players and displays success message.
- If invalid: system rejects and provides actionable feedback.

4.1.3 Functional Requirements

- REQ-1: When the user uploads a CSV, the system shall validate required fields.
- REQ-2: If required fields are missing, the system shall provide a detailed error message.
- REQ-3: When the CSV passes validation, the system shall write all records to the MongoDB cluster.
- REQ-4: If the upload is interrupted, the system shall allow the user to retry without reselecting the file.
- REQ-5: if duplicate players appear, the system shall prompt the user to replace entries.

4.2 Real-Time Team Communication

4.2.1 Description and Priority

Provides real-time chat between authenticated users via WebSockets.

Priority: Medium

4.2.2 Stimulus/Response Sequences

- User sends message: Message is broadcast to all participants.
- User joins chat: Previous messages are displayed when available.

4.2.3 Functional Requirements

- REQ-6: When a user sends a message, the system shall broadcast it to all active chat participants.

- REQ-7: If the WebSocket connection fails, the system shall attempt reconnection every 2 seconds for 30 seconds.
- REQ-8: When reconnection succeeds, the system shall fetch missed messages from Redis history.
- REQ-9: If authentication fails, the system shall deny chat access and display an error.

4.3 Real-Time Match Event Logging

4.3.1 Description and Priority

Analysts can log match events in real time and sync them across devices.

Priority: High

4.3.2 Stimulus/Response Sequences

- User taps event button: Event logged and synced across devices.

4.3.3 Functional Requirements

- REQ-10: When an analyst selects an event type, the system shall attach the current match timestamp.
- REQ-11: When an event is logged, the system shall sync it to all connected manager dashboards within 300ms.
- REQ-12: If the network is offline, the system shall store events locally and resync on reconnection.
- REQ-13: If event saving fails, the system shall notify the analyst and allow immediate retry.

4.4 Automated Performance Feedback (Analytics Engine)

4.4.1 Description and Priority

Computes insights, warnings, and recommendations based on match data.

Priority: High

4.4.2 Stimulus/Response Sequences

- User opens analytics panel: System loads data and computes insights.
- New match data recorded: System recalculates recommendations.

4.4.3 Functional Requirements

- REQ-14: When the manager opens the analytics panel, the system shall compute and display metrics.
- REQ-15: When new events arrive, the system shall recalculate KPIs incrementally.
- REQ-16: If insight generation fails, the system shall fall back to basic metrics.
- REQ-17: If the analytics engine times out after 3 seconds, the system shall show a loading error with a retry option

4.5 Video Timestamp Jumping & Clip Extraction

4.5.1 Description and Priority

Allows users to upload videos and jump directly to logged event timestamps.

Priority: High

4.5.2 Stimulus/Response Sequences

- User uploads video: System processes metadata and prepares playback.
- User selects event: Video jumps to matching timestamp.

4.5.3 Functional Requirements

- REQ-18: When the user uploads a match video, the system shall extract metadata.
- REQ-19: When a logged event is selected, the system shall move the video to the event timestamp.
- REQ-20: If a timestamp is invalid, the system shall show an error.
- REQ-21: If video upload exceeds 500 MB, the system shall warn the user and request confirmation.

4.6 User Onboarding & Interactive Tutorial

4.6.1 Description and Priority

Provides a guided tutorial for first-time users.

Priority: Medium

4.6.2 Stimulus/Response Sequences

- First-time user opens app: Tutorial appears automatically.
- User accepts tutorial: Walkthrough begins.
- User skips tutorial: Becomes available in Help menu.

4.6.3 Functional Requirements

- REQ-22: When a first-time user opens the app, the system shall/ show a guided walkthrough.
- REQ-23: When the user skips the tutorial, the system shall/ store this preference for future sessions.
- REQ-24: If the interactive tutorial fails to load, the system shall show a text-only version.
- REQ-25: When the tutorial is completed, the system shall unlock the full interface.

4.7 Cockburn use cases

Import CSV Player Data		
Goal in Context	Load player roster quickly via spreadsheet upload for immediate use in match analytics.	
Preconditions	<ul style="list-style-type: none">● CSV file follows required format.● Analyst authenticated.● File size under 2MB limit.	
Success End	Players validated, stored, available for match logging	
Failed End	File rejected with error details displayed	
Primary Actor	Analyst	
Trigger	Analyst taps "Import CSV" button	
Step	Analyst Action	System Response
1	Tap upload button	Show file picker dialog
2	Select CSV file	Validate structure and fields
3	Confirm import	Save players to database
4	Wait for processing	Show success message, refresh list
Extensions	2a. Invalid format → Show specific column errors 3a. Duplicate players → Offer merge or skip options	

Real-Time Team Chat		
Goal in Context	Enable instant communication between analyst and manager about match events and tactical insights.	
Preconditions	<ul style="list-style-type: none">Both users authenticated and in same match sessionWebSocket connection activeMatch is active	
Success End	Message delivered and visible to all participants	
Failed End	Message queued locally with "sending..." indicator	
Primary Actor	Analyst or Manager/Coach	
Trigger	User taps "Send" after typing message	
Step	User Action	System Response
1	Type message	Enable send button
2	Tap send	Validate authentication token
3	Wait for delivery	Store message, broadcast to all
4	See confirmation	Show "delivered" status
Extensions	2a. Token expired → Prompt re-authentication 3a. Offline → Queue message, auto-send when connected	

Log Match Event	
Goal in Context	Record granular match actions with player, location, and time for real-time analytics and post-match review.

Log Match Event		
Preconditions	<ul style="list-style-type: none">● Match is active and timer running● Analyst authenticated● Selected player is in match roster	
Success End	Event stored, synced to all devices, charts updated	
Failed End	Event queued locally, sync pending indicator shown	
Primary Actor	Analyst	
Trigger	Analyst taps event type button (e.g., "Goal", "Key Pass")	
Step	Analyst Action	System Response
1	Tap Event Button	Show player selection list
2	Select player	Show pitch zone grid
3	Tap pitch zone	Attach timestamp, record event
4	Confirm	Sync to all dashboards within 300ms
Extensions	2a. Player not listed → Quick-add dialog 4a. Network offline → Store locally, auto-sync on reconnect	

View Real-Time Dashboard	
Goal in Context	Access live match statistics and visualizations to support immediate tactical decisions.
Preconditions	<ul style="list-style-type: none">● Manager authenticated● Match is actively being logged● Dashboard device has stable connection
Success End	Dashboard loads with current stats and live updates
Failed End	Show cached data with "connection lost" warning
Primary Actor	Manager/Coach

View Real-Time Dashboard		
Trigger	Manager opens match from active list	
Step	Manager Action	System Response
1	Open match dashboard	Load initial stats via REST API
2	Wait for data	Render charts and graphs
3	View screen	Subscribe to WebSocket updates
4	Watch live	Receive real-time event updates
Extensions	1a. Match not active → Show "Awaiting Data" state 3a. Network drops → Show offline mode indicator	

Jump to Video Timestamp		
Goal in Context	Review specific match events by jumping directly to the corresponding moment in the uploaded video.	
Preconditions	<ul style="list-style-type: none">● Video file uploaded for current match● Event has valid timestamp● Manager authenticated	
Success End	Video seeks to correct time and begins playback	
Failed End	Show error, offer manual seek or timestamp correction	
Primary Actor	Manager/Coach	
Trigger	Manager taps event in timeline	
Step	Manager Action	System Response
1	Fetch match events	Retrieve from database
2	Compute metrics	Calculate KPIs and trends
3	Generate insight	Create recommendations
4	Push to dashboard	Update charts within 3 seconds

Jump to Video Timestamp	
Extensions	2a. Timeout after 3s → Show fallback metrics 4a. Computation fails → Display basic stats only

Run Analytics Engine		
Goal in Context	Compute performance insights, warnings, and recommendations from match data automatically.	
Preconditions	<ul style="list-style-type: none">• Sufficient match events logged (minimum 5)• Analytics engine is running• Database connection stable	
Success End	Insights generated and displayed on manager dashboard	
Failed End	Show fallback basic stats with error notification	
Primary Actor	System (triggered by Analyst or Manager)	
Trigger	Manager opens analytics panel OR new event logged	
Step	System Action	Response
1	Tap video clip icon	Fetch event timestamps from database
2	Select specific event	Calculate seek time (event - 5s)
3	Wait for seek	Video player jumps to timestamp
4	Watch playback	Highlight event in timeline
Extensions	2a. No timestamp → Show "link unavailable" message 3a. Video not ready → Queue seek command	

Complete Interactive Tutorial		
Goal in Context	Guide first-time users through key features to reduce learning curve and improve adoption.	
Preconditions	<ul style="list-style-type: none"> • User launching app for first time • No prior onboarding completion recorded • Device meets minimum requirements 	
Success End	Tutorial completed, user can access full functionality	
Failed End	Show text-only help, flag for support contact	
Primary Actor	New User (Analyst or Manager)	
Trigger	App detects no onboarding flag on launch	
Step	User Action	System Response
1	Launch app	Detect first-time status
2	Tap "Start Tutorial"	Show overlay highlighting feature 1
3	Tap "Next"	Show overlay highlighting feature 2
4	Tap "Skip to End"	Set onboarding flag = complete
5	Return to main screen	Show full app interface
Extensions	2a. Interactive fails → Show static text version 3a. User exits → Save progress, resume next time	

5. Other Nonfunctional Requirements

5.1 Performance Requirements

P-1: Event synchronisation must have an end-to-end delay of under 300 milliseconds so that match updates appear almost instantly on all connected devices.

P-2: Analytics panels must load within 2 seconds during normal use so the system stays responsive.

P-3: The system must be able to handle up to 10,000 logged match events in a single match without slowing down or affecting the interface.

P-4: The application must stay fully responsive with at least 50 users online at the same time, as this reflects the expected load during a live match.

P-5: Video files up to 500 MB must upload, process, and become ready for playback within 30 seconds to allow quick review at half-time or after the match.

5.2 Safety Requirements

S-1: Any events recorded while offline must stay stored on the device until the system reconnects and successfully syncs the data.

S-2: Actions such as deleting match data or changing player information must require the user to confirm the action before it goes ahead.

S-3: The system must handle Redis connection loss safely and try to reconnect so that real-time communication and analytics start working again as soon as the connection returns.

5.3 Security Requirements

SEC-1: All communication between the client, the server, and cloud services must be encrypted so that sensitive match data cannot be intercepted.

SEC-2: User login must use JWT tokens that expire every 24 hours, with a secure method for refreshing tokens to keep sessions active.

SEC-3: Role-based access must stop coaches from editing raw match events so that the data stays accurate and unchanged.

SEC-4: Sensitive information, including user details, performance data, or authentication tokens, must not be stored in local storage to prevent unauthorised access.

5.4 Software Quality Attributes

Q-1: The system should aim to be available at least 99% of the time during matches so that users can rely on it throughout the game.

Q-2: Analysts must be able to log standard events in two taps or fewer, allowing quick use during fast-paced situations.

Q-3: The system must support up to 10 matches being analysed at the same time without needing any changes to the system design.

Q-4: The codebase must use a modular structure so the front end, back end, analytics engine, and communication system can be maintained and updated independently.

5.5 Business Rules

BR-1: Only authenticated analysts can log match events, while authenticated coaches can view tactical dashboards and insights but cannot record events.

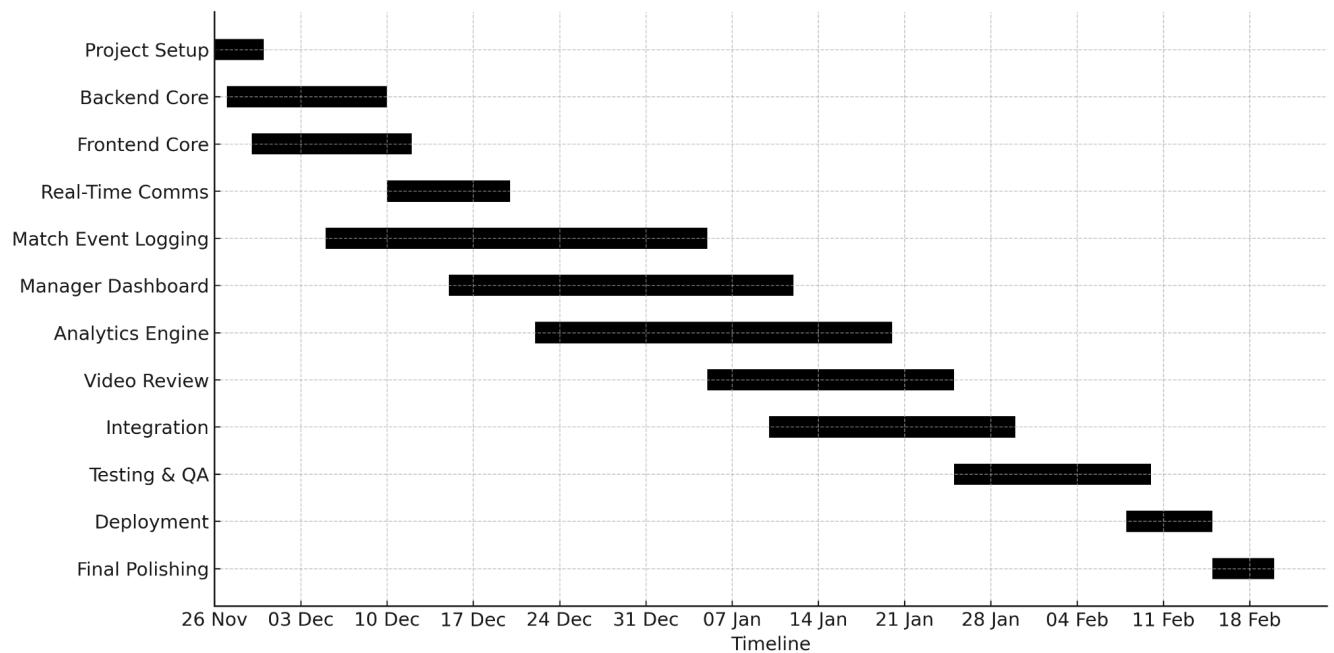
BR-2: Only analysts can edit player performance data, while coaches can view but not modify raw match logs.

BR-3: All match events and performance information must be linked to a specific team and competition so that season-wide analytics remain consistent and organised.

BR-4: Any uploaded video file must match the correct match ID so that timestamps line up accurately during playback.

6. Timeline

Phase	Description	Start	End	Dependencies
Project Setup & Environment Configuration	Repo setup, project structure, CI/CD setup	26 Nov	30 Nov	None
Backend Core Setup	Django project, user models, login, JWT/session	27 Nov	10 Dec	1
Frontend Core Setup	Navigation, layout system, authentication UI	29 Nov	12 Dec	1
Real-Time Communication	WebSocket server, channels, chat prototype	10 Dec	20 Dec	2
Match Event Logging Module	Event buttons, timestamping, offline syncing	5 Dec	5 Jan	2, 3
Manager Dashboard Module	Stats UI, charts, team view, player metrics	15 Dec	12 Jan	3, 5
Analytics Engine Development	Heatmaps, performance metrics, pattern detection	22 Dec	20 Jan	5
Video Review Module	Video upload, event timestamp linking, scrubbing	5 Jan	25 Jan	5
Integration & Syncing	Sync between analyst–manager views, testing flows	10 Jan	30 Jan	4, 5, 6, 7, 8
System Testing & QA	User testing, load testing, bug fixing	25 Jan	10 Feb	9
Deployment & Final Packaging	Cloud hosting, build signing, app packaging	8 Feb	15 Feb	10
Final Polishing & Submission	UI polish, documentation, demo prep	15 Feb	20 Feb	11



7. Other Requirements

Appendix A:

Analyst

A user responsible for recording match events, managing data quality, and interpreting analytics.

Manager / Team Manager

A user responsible for tactical decisions, team selection, and reviewing high-level performance insights.

Coach

A user who uses the system to monitor player and team performance, typically during training and matches.

Event (Match Event)

A discrete action recorded during a match (e.g. *shot*, *key pass*, *duel won*, *possession lost*, *error*), associated with a player, team, pitch zone, and timestamp.

Timestamp

A time marker associated with a match event, aligned with the official match clock (e.g. 37:12 in the first half).

Pitch Zone

A defined area of the football pitch used to spatially classify events (e.g. defensive third, central midfield, right wing).

Dashboard

A screen that presents real-time and historical statistics, charts, and insights for managers and coaches.

Analytics Engine

The part of the system that processes raw match data into metrics, trends, and performance recommendations.

Match ID

A unique identifier representing a single match within the system.

User Role

A classification of user permissions (e.g. Analyst, Manager, Coach) that determines available features and data access.

WebSocket

A protocol that enables full-duplex real-time communication between the client and server.

JWT (JSON Web Token)

A token format used for securely representing authenticated user identity and permissions.

MongoDB Atlas

A managed cloud database service used as the primary persistence layer for StatSync.

Redis

An in-memory data store used for message brokering and real-time communication coordination.

Appendix B:

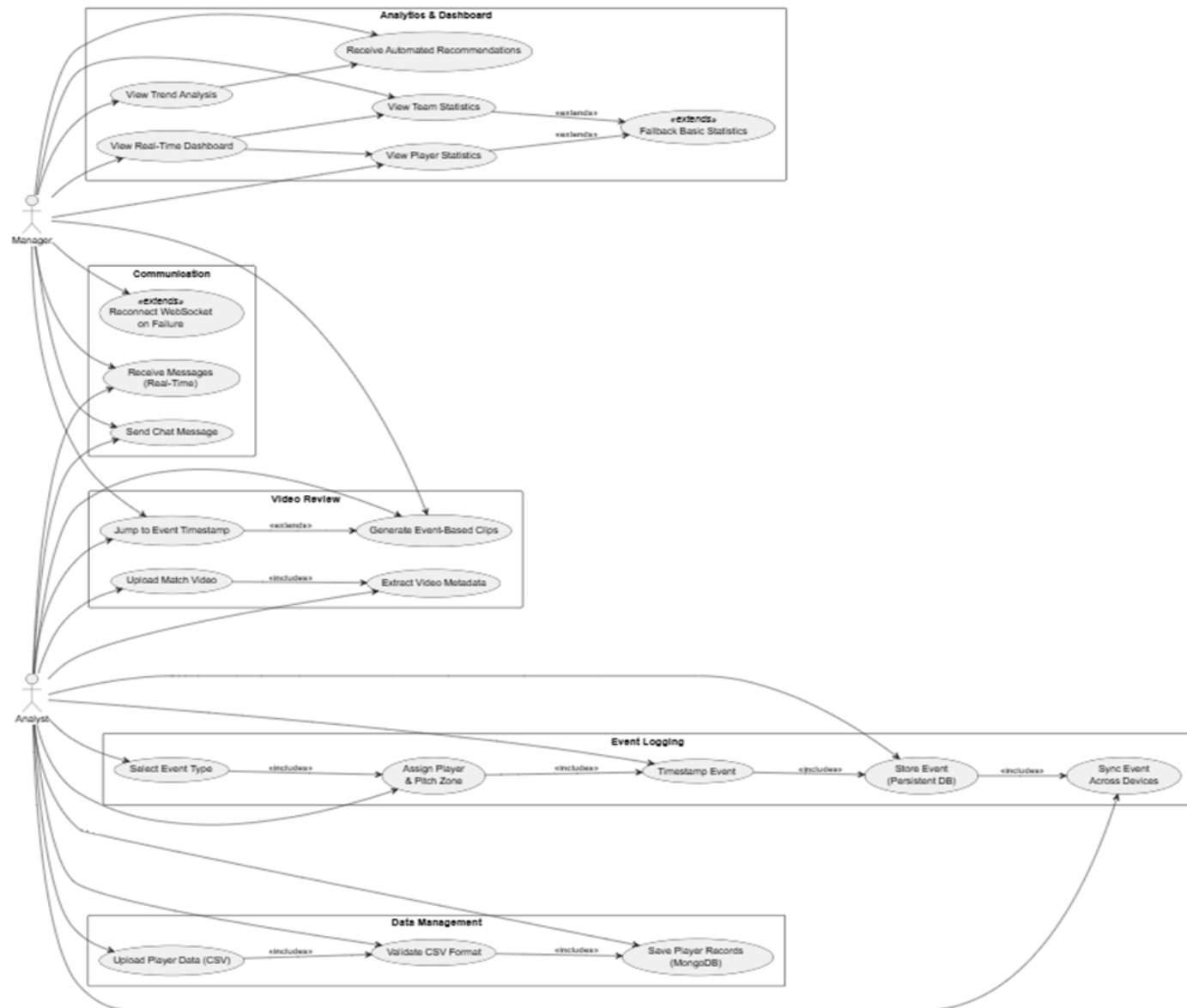


Figure B.1 – Use Case Diagram (Analyst & Manager) This diagram illustrates the main interactions between the Manager and Analyst users with the StatSync system. It shows the main functionalities including event logging, analytics dashboards, video review, and real-time communication.

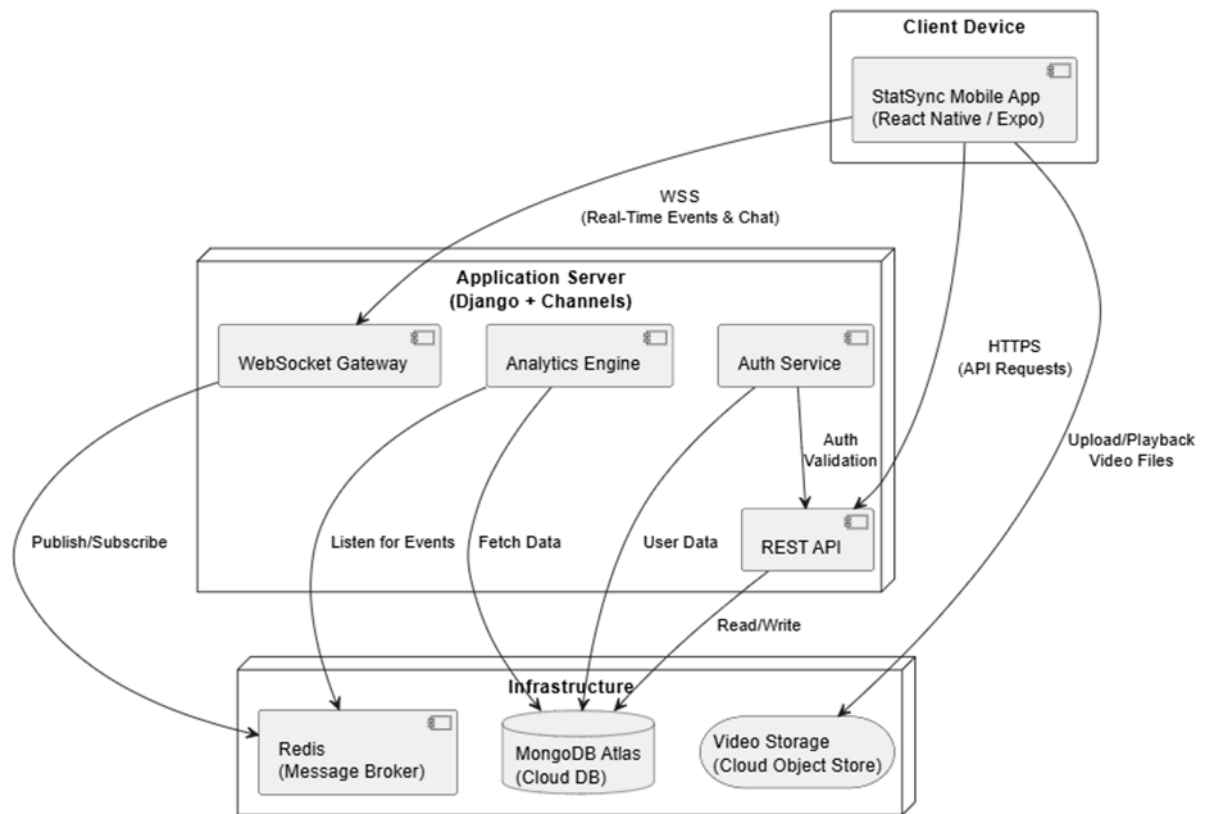


Figure B.2 – Component Diagram

This component diagram presents the high-level architecture of StatSync, showing how the app communicates with the Django backend, WebSockets, and cloud database

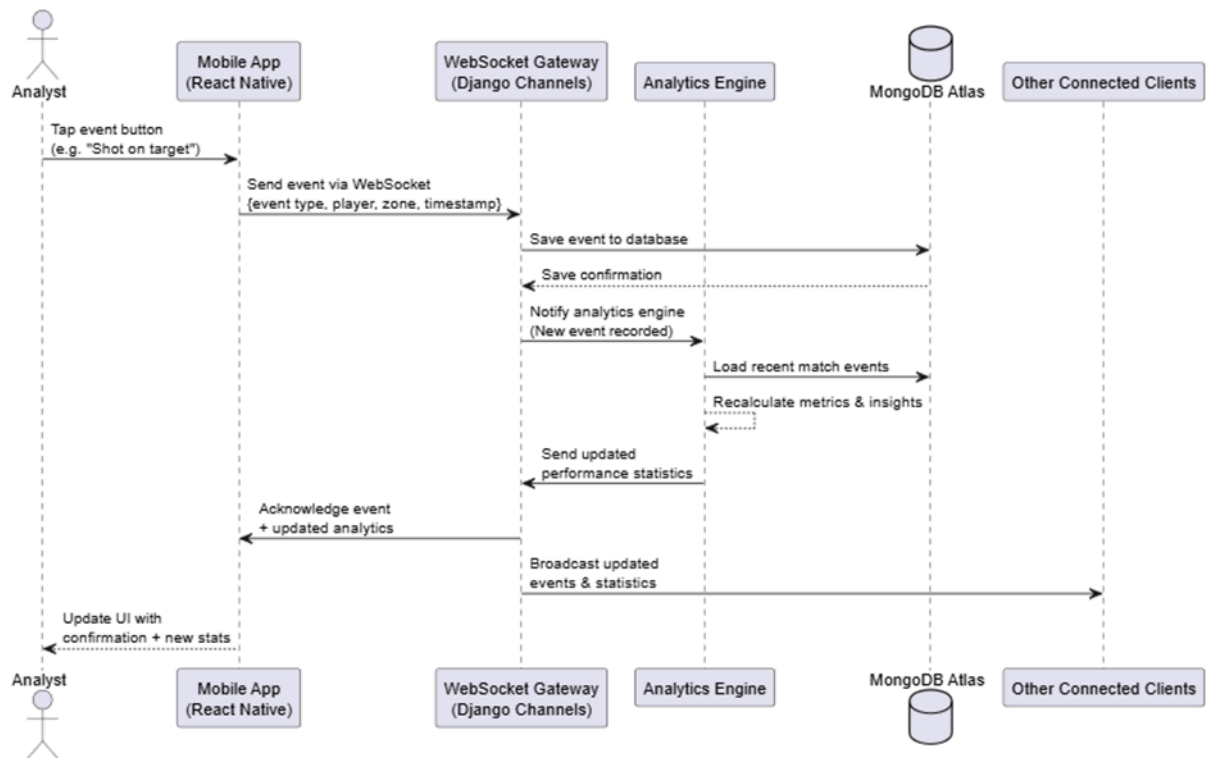


Figure B.3 – Sequence Diagram: Analyst Logs Event

This sequence diagram describes when an Analyst records an event. The mobile app sends the event through the WebSocket gateway, the server stores it in the database, and the analytics engine calculates metrics before updating all clients connected.

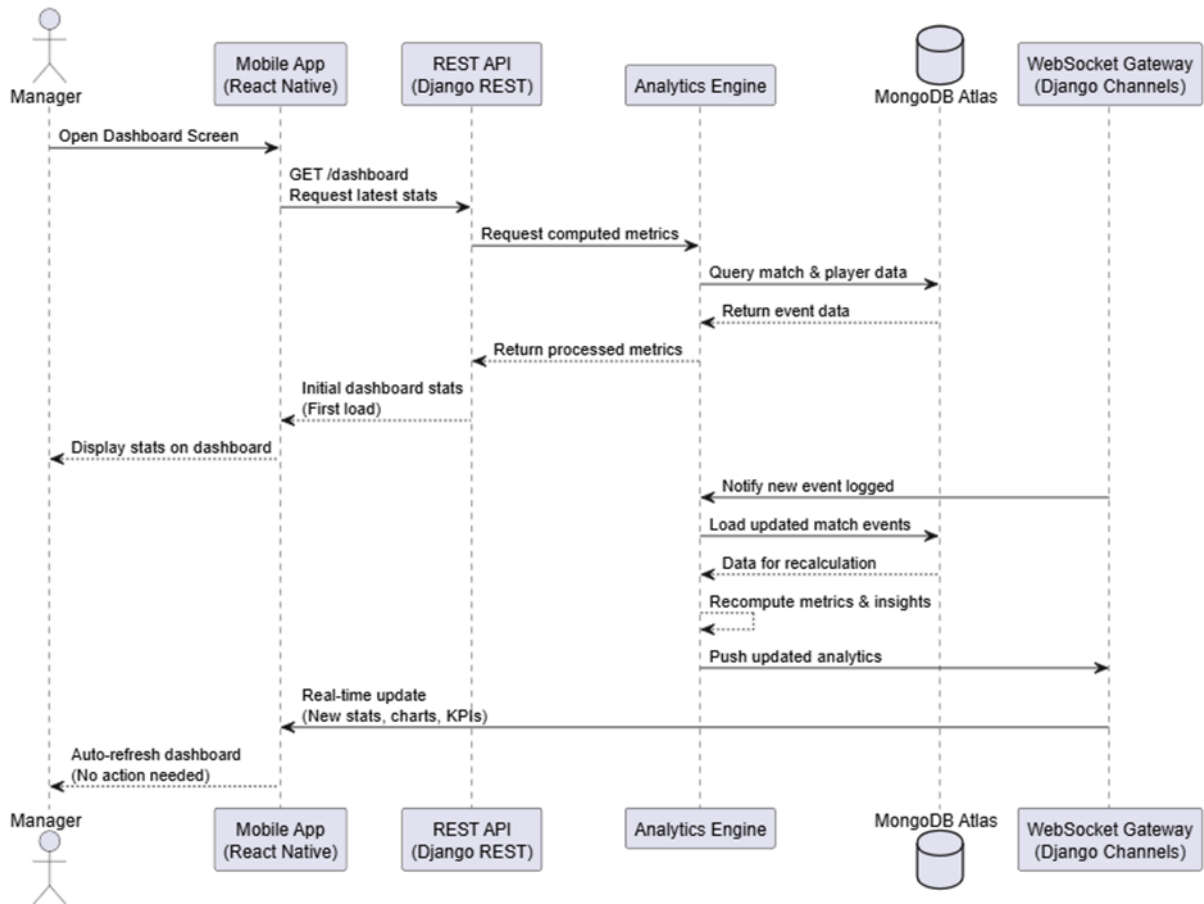


Figure B.4 – Sequence Diagram: Manager Views Dashboard

This diagram shows the process when a Manager opens the dashboard. Initial statistics are loaded through the REST API, and updates are sent automatically to the mobile app via WebSockets.

Appendix C:

Event Schema

```
{  
  "eventId": "string",  
  "matchId": "string",  
  "playerId": "string",  
  "timestamp": "mm:ss",  
  "eventType": "string",  
  "zoneId": "string",  
  "createdAt": "ISODate"  
}
```

Match Schema

```
{  
  "matchId": "string",  
  "homeTeam": "string",  
  "opponentTeam": "string",  
  "date": "ISODate",  
  "teamId": "string",  
  "events": ["eventId"],  
  "videoFile": "url"  
}
```

User Schema

```
{  
  "userId": "string",  
  "role": "analyst | coach",  
  "teamId": "string",  
}
```

Chat Message Schema

```
{  
  "messageId": "string",  
  "senderId": "string",  
  "matchId": "string",  
  "timestamp": "ISODate",  
  "content": "string"  
}
```