System Requirements Specifications
UPDATED SOFTWARE REQUIREMENT SPECIFICATION

Introduction

The need for **AEGIS** arises from significant challenges in current digital forensics workflows, where teams face limitations in secure collaboration, effective evidence management, and maintaining a verifiable chain of custody. These gaps hinder the integrity, efficiency, and legal defensibility of investigations.

AEGIS aims to address these issues by delivering a robust, containerized platform that empowers cybersecurity professionals to collaborate securely in real time, manage digital evidence systematically, and ensure full compliance with legal and forensic standards. The platform will support end-to-end investigation management, emphasizing automation, integrity, and accountability across all stages of the digital forensics process.

USER STORIES

User stories represent functional goals from the perspective of each user role. They illustrate how users interact with the system to accomplish specific tasks, such as uploading evidence, creating annotation threads, sharing cases, or approving findings. These stories guide feature prioritization and serve as acceptance criteria for development.

1. Use Case 1: Real-Time Collaboration: Secure Chat

US1: As an **admin**, I want to be able to create a chat room for every case and add the assigned members to that case.

US2: As a **DFIR team member**, I want to chat in real-time with other analysts or team members assigned to the same case, so that we can collaboratively investigate and resolve incidents efficiently without leaking information.

US3: As an **External Collaborator**, I want secure communication channels so that I can provide expertise and receive case updates without compromising confidentiality.

Acceptance Criteria:

For an admin

- An admin must be able to create a chat room.
- An admin must be able to delete a chat room.
- An admin must be able to update a chat room (*??)
- An admin must be able to add DFIR team members to their respective chat rooms.

- An admin must be able to remove DFIR team members to their respective chat rooms.
- An admin should be able to view all chat rooms.
- An admin must be able to view all users added to a chat room.

<u>For a DFIR team member</u>

- A DFIR team member must be added to a case-specific chat room only if they are assigned to the case.

<u>For an External collaborator</u>

- External Collaborators must be able to join a case via a link shared by an administrator.
- An external collaborator must have access to a chat for a limited amount of time based on a token that expires after a set period of time.

<u>For all chat members</u>

- All messages sent by chat members must be end-to-end encrypted.
- All chat participants must be able to send and receive real-time updates and notifications.
- Plain text messages and evidence attachments must be supported.
- Chat members should be able to read all the messages in the chat.
- Chat members must be able to delete their own messages

2. Use Case 2: Real-Time Collaboration: Discussion Threads

US1: As a **DFIR Team Member**, I want to create and discuss detailed annotation threads on specific evidence files so that I can document my analysis, track insights or discrepancies, and collaborate effectively with other assignees on complex findings.

US2:**As a Lead DFIR Investigator**, I want to review and manage annotation threads so that I can ensure analysis quality and coordinate team efforts.

US3: **As an External Collaborator**, I want to contribute distinctly identifiable annotations so my specialised input is visible while maintaining evidence integrity.

Acceptance Criteria:

<u>For a DFIR Team member and any member with access to a case</u>

- A DFIR team member must be able to create annotation threads on specific evidence files.
- A DFIR team member must be able to reply to, edit, and delete their own annotations.

- A DFIR team member must be able to tag collaborators in the annotation threads.
- DFIR team members must be able to view all annotations related to their assigned cases.
- A DFIR team member can approve (or react to) all annotations.

For a Lead DFIR Investigator

- A lead DFIR investigator can additionally delete annotations created by other DFIR team members.
- Can approve or reject annotations for final review.
- A lead DFIR investigator should be able to add a case collaborator to participate in the thread.
- A lead investigator should be able to update a thread's priority.

For an External Collaborator

- Annotations that are made by external collaborators must be marked with their role, “External Collaborator,” for easy identification.

3. Use Case 3: Automated Logging

US1: As a **compliance officer**, I want all critical actions to be logged and immutable so that I can verify the integrity of the investigation and enforce chain-of-custody

US2: As a **DFIR Administrator**, I want to be able to access and view comprehensive automated logging of all system activities so that I can monitor team progress and ensure proper evidence handling procedures.

US3:As a **DFIR Team Member**, I want my analysis activities to be automatically logged so that I can focus on investigation work without manual documentation overhead.

Acceptance Criteria:
For a Compliance Officer

- All sensitive actions (e.g., logins, uploads, deletions, approvals) must be automatically logged.
- Each log entry must include a timestamp, action, actor (user ID) and IP address.
- Each log entry must be tamper-proof and cannot be altered post-creation.
- All logs must be retained and stored for legal and compliance auditing.

For an administrator

- All user actions must be automatically logged and must include a timestamp, action, actor (user ID) and IP address.

- An admin must be able to view logs and filter them depending on the user, case, date or the type of action performed.
- An admin must be able to download or export logs for reporting and auditing purposes.

For a DFIR team member

- All actions performed by a DFIR team member such as viewing evidence, interactions in the annotations must be logged automatically.
- Every team member must be able to view their own action history or logs.

USER CHARACTERISTICS

This section describes the profiles, skill levels, and roles of the users who will interact with the AEGIS platform. It covers system administrators, forensic investigators, external collaborators, and auditors—each with varying technical proficiency and access needs. These insights help shape user interfaces and access control logic.

1. DFIR Administrator

Role Description: Primary administrator responsible for overall case management, team coordination, and platform governance.

User Characteristics:

- Digital forensics professional responsible for case management, team coordination, and platform governance. They handle case allocation, resource management, and maintain evidence integrity while requiring comprehensive oversight capabilities and administrative privileges for audit compliance and security monitoring.
- Can review all evidence maps and timelines.
- Can access and export reports for audit or governance purposes.
- Maintains overall administrative oversight across new collaboration and visualization tools.

2. Lead DFIR Investigator

Role Description: Investigators who lead case analysis, coordinate team activities, and make critical investigative decisions.

User Characteristics:

- Digital forensics specialists with team leadership responsibilities who coordinate evidence analysis workflows and make critical investigative decisions. They facilitate team discussions, perform quality control on findings, and serve as the primary communication hub for case updates.
- Manages and approves annotation threads and case timelines.
- Can edit, organize, and highlight evidence maps.

- Reviews and customizes case reports for legal or organizational standards.

3. DFIR Team Members (Investigators)

Role Description: Hands-on investigators who conduct detailed analysis of digital evidence and contribute to case resolution.

User Characteristics:

- Digital forensics specialists who conduct hands-on analysis of digital evidence. They require collaborative tools for real-time coordination, spend most of their time examining and annotating evidence, and create comprehensive documentation throughout the investigation process.
- Conducts detailed analysis, creates annotations, and links evidence to timeline events.
- Uses graphical evidence maps to visualize connections and patterns.
- Contributes to reports, ensuring their analysis is accurately captured.

4. External Collaborators

Role Description: External professionals who provide specialised assistance on specific cases with limited system access.

User Characteristics:

- External professionals including legal counsel, law enforcement officers, or specialised consultants who provide expertise on specific cases. They require temporary, restricted access to assigned cases and secure communication channels while maintaining clear audit trails of their contributions for legal purposes.
- Can view graphical maps and timelines with restricted editing access if authorized.
- Contributions must be clearly tagged in annotations and reports.
- Maintains secure communication and audit visibility.

FUNCTIONAL REQUIREMENTS

This section details the specific functional capabilities the AEGIS system must provide to fulfill user needs. Each requirement is derived from use cases, user stories, and operational needs, and is structured to define what the system must do, including evidence uploads, chain of custody tracking, user management, and role-based collaboration.

R1: User Registration and Authentication

This section defines how users will be able to securely gain access to the platform including account creation and registration, secure credential verification, session management and logout.

R1.1:User Registration

R1.1.1: Registration of a New User
The system shall allow an admin to register a new user by providing their full name, email address, and a randomly generated password.
R1.1.2: Email Format and Uniqueness Validation
The system must validate the format of the submitted email and ensure it is not already associated with an existing user.
R1.1.3: Role Assignment
The system must allow an admin to assign a specific role from a list of predefined roles during registration.

R1.2: Login and Credential Verification

R1.2.1: User Login
The system must authenticate users by verifying their email and password, issuing a secure session token upon success.
R1.2.2: Email Verification
The system must send a time-sensitive verification token to a newly registered email, which the user must activate through verification in order to gain access to the platform.
R1.2.3: Credential Security
The system shall store user passwords using secure, one-way hashing (bcrypt) to prevent credential theft.
R1.2.4: Failed Log In
The system must implement rate limiting and account lockout mechanisms after repeated failed login attempts.

R1.3: Session and Token Management

R1.3.1: Session Token Expiry
The system shall ensure that issued tokens expire after a configurable period of inactivity.
R1.3.2: Reconnection and Token Refresh
The system should automatically refresh session tokens or redirect the user to reauthenticate upon expiration.
R1.3.3: Logout
The system must allow users to log out manually, which just revoke their current session token and mark the session inactive.

R2: Role & Permission Management (RBAC)

This section defines how the system manages roles, permissions, and access control using a Role-Based Access Control (RBAC) strategy. It outlines how permissions are enforced, roles are assigned, and security is maintained throughout the application.

R2.1: Role Definition and Storage

R2.1.1: Predefined Roles
The system shall define and store a set of predefined roles (e.g., Admin, Investigator, Responder, Analyst) using an ENUM-based schema in the database.
R2.1.3: Access control
Each role shall represent a set of responsibilities and access boundaries defined by associated permissions.

R2.2: Permission Definition and Association

R2.2.1: Permissions that map to actions
The system shall define a comprehensive list of granular permissions (e.g., `CREATE_CASE`, `VIEW_EVIDENCE`, `ASSIGN_ROLE_TO_USER`), which reflect all actionable operations in the system.
R2.2.2: Permission & Role Association
Permissions shall be stored in a dedicated table and shall be associated with one or more roles via a many-to-many relationship.
R2.2.3: Hierarchical Permission Inheritance
The system shall support hierarchical permission inheritance if required (e.g., Admin implicitly has all permissions granted to other roles).

R2.3: Permission Enforcement

R2.3.1: Access Control Authorization
The system shall enforce access control at the API level using middleware that verifies whether the authenticated user has the required permission for the requested action.
R2.3.2: Unauthorized Attempt Error Response & Audit Logging
Unauthorized access attempts shall result in appropriate HTTP error responses (e.g., 403 Forbidden) with error logging for audit purposes.
R2.3.3: Centralised Permissions Check
Permission checks shall be centralized and reusable to ensure consistent enforcement across all services and endpoints.

R2.4: Role Assignment Scope

R2.4.1: Platform-Wide Global Roles & Case-Specific Roles
The system shall support **global roles**, which apply platform-wide, and **case-specific roles**, which grant permissions limited to a particular case context.
R2.4.2: Role/User Mapping
The system shall allow mapping users to roles per case (e.g., an investigator may be assigned to `Case A` only, with restricted visibility to `Case B`).

R2.4.3: Role Enforcement
Case-level roles shall be enforced using contextual authorization checks during case-related operations.

R2.5: Administrative Role Management

R2.5.1: System & Case-Level Role Viewing
The system shall allow administrators to view a list of all users along with their assigned global and case-level roles.
R2.5.2: Updating User Roles
The system shall allow administrators to assign or update a user's role through a secure interface or API, with appropriate logging of the change.
R2.5.3: Case Assignment & Role Designation
The system shall allow only admins to assign users to a specific case and designate a role (e.g., `Investigator`, `Responder`) for each.

R3: Case Management

This section outlines the functional requirements for creating, managing, and updating investigation cases within the system. It includes functionality related to case lifecycle, access, team assignment, and operational metadata such as status, priority, and investigation stage.

R3.1: Case Creation

R3.1.1: Role Case Creation
The system shall allow only admins to create a new case by providing a title, description, their name, and a team name.
R3.1.2: Case ID
The system shall generate a unique identifier (e.g., UUID) for each case upon creation and store a timestamp of creation.
R3.1.3: Audit Log Case Creation
The system shall log every case creation event, including:

- User ID of the creator
- Case ID and title
- Timestamp of creation
- Initial values (description, team name, owner)

R3.2: Case Viewing

R3.2.1: Assigned Case Viewing
The system shall allow users to view their assigned cases.
R3.2.2: Filtering and Sorting
The case listing shall support filtering and sorting by fields such as status, priority, creation date, or investigation stage.
R3.2.3: Case Details & Assigned Users Viewing
The system shall allow viewing of individual case details, including assigned users,

status, priority, timeline history, and metadata.

R3.2.4: Audit Logging of Case Viewing

The system shall log each case view action, each log will include the following mandatory fields, and any other attributes deemed necessary for specific actions:

- User ID of the viewer
- Case ID viewed
- Timestamp of access

R3.3: Case Status Updates

R3.3.1: Authorized Case Update Access

The system shall allow authorized users with the `UPDATE_CASE_STATUS` permission to update the status of a case to one of the predefined states (e.g., `open`, `under_review`, `closed`).

R3.3.2: Audit Logging of Case Status Updates

Case status transitions shall be logged with:

- Case ID
- Acting user ID
- Previous and new status values
- Timestamp of change

R3.3.3: Restrictive Case Status Update

The system shall enforce any defined status transition rules (e.g., disallow reopening a closed case without elevated permissions).

R3.4: Case Role Assignment

R3.4.1: Permission

Assigned roles shall define the user's permissions within that specific case context, as enforced by the RBAC system.

R3.4.2: Updating and Revoking Case-Level Roles

The system shall support updating and revoking case-level role assignments.

R3.4.3: Audit Logging of Case-Level Role Assignments and Updates

The system shall log all case-level role assignments or updates, including:

- Acting user ID
- Target user ID
- Case ID
- Assigned or revoked role
- Timestamp of the action

R3.5: Case Metadata Updates

R3.5.1: Authorized Users Can Update Case Metadata

The system shall allow authorised users with the `UPDATE_CASE_METADATA` permission to update the **priority** of a case (e.g., `low`, `medium`, `high`, `critical`).

R3.5.2: Investigation Stage Updates By Authorized Users

The system shall allow authorized users to update the **investigation stage** of a case (e.g., `analysis`, `research`, `evaluation`, `finalization`), used for tracking case progress.

R3.5.3: Audit Logging for Investigation Stage Updates

The system shall log all changes to case metadata (priority or stage), including:

- Case ID
- Acting user ID
- Field modified (e.g., priority or stage)
- Old and new values
- Timestamp of change

R4: Evidence Management

This section outlines the requirements the system shall fulfill for the secure and careful management of digital evidence, including file upload to IPFS, metadata storage, retrieval, deletion, and integrity verification.

The system shall ensure that all critical actions are logged for accountability and an immutable chain of custody.

R4.1: Uploading Evidence

R4.1.1: Evidence Upload By the Admin

The system shall allow only the system admin to upload evidence. This is so as to ensure fine-grained control over the handling of evidence, given the sensitivity of evidence uploaded to the platform.

R4.1.2: Uploaded Evidence Stored on IPFS

Upon successful upload, evidence files shall be stored on IPFS, and their CIDs (Content Identifiers) shall be returned and stored in PostgreSQL to allow the retrieval of evidence.

R4.1.3: Audit Logging For Uploaded Evidence

The system shall log each evidence upload and include the following attributes:

- User ID of the uploader
- Case ID (where applicable)
- Filename and file type
- IPFS CID and file size
- Timestamp of upload

R4.2: Checksum Generation

R4.2.1: Computed Checksum for Each Uploaded File

The system shall compute a cryptographic checksum (e.g., SHA-256) of each uploaded file at the time of upload.

R4.2.2: Checksum Stored as Part of Metadata

The checksum shall be stored in the evidence metadata for future validation.

R4.2.3: Audit Log Computed Checksum
The system shall log the computed checksum along with the evidence upload event (covered in R4.1.3).

R4.3: Metadata Storage

R4.3.1: Metadata Stored Fields
The system shall store the following metadata for each evidence file:

- IPFS CID
- File size (in bytes)
- File type (MIME)
- Original filename
- Upload timestamp
- Uploader’s user ID
- Associated case ID
- Checksum value

R4.3.2: Metadata Stored In PostgresQL
Metadata shall be stored in a structured relational database (e.g., PostgreSQL) and linked to the case and user entities.

R4.4: Evidence Retrieval

R4.4.1: Case Level, Evidence Metadata Retrieval
The system shall allow users which are assigned to a case to retrieve evidence metadata and details of all the evidence uploaded for that case by:

- Evidence ID
- Case ID

R4.4.2: Audit Logging for Evidence Access
The system shall log every evidence access event. Each log entry shall include the following mandatory fields:

- User ID of the requester
- Evidence ID or Case ID
- Timestamp of access
- IP address (optional for forensics)

R4.5: Evidence Deletion

R4.5.1: Evidence Deletion Authorization
The system shall allow only the system admin to delete evidence.

R4.5.2: Mark Evidence for Deletion
Deleted evidence shall be soft-deleted (flagged) to preserve auditability unless a hard delete is explicitly authorized by an admin.

R4.5.3: Audit Logging for Evidence Deletion

The system shall log each deletion action. Each log entry shall include the following mandatory fields:

- User ID performing deletion
- Evidence ID
- Case ID
- Timestamp of deletion
- Reason for deletion (optional comment field)

R4.6: Checksum Validation

R4.6.1: Authorization for Evidence Integrity Check
The system shall allow system admins to validate the integrity of evidence files by re-computing and comparing its checksum.
R4.6.2: Compromised Evidence
The system shall determine and display the outcome of the checksum validation by comparing the current hash of the evidence file with its originally stored checksum. If the values differ, the evidence shall be flagged as compromised; if they match, it shall be marked as intact. This result must be visibly presented to the validating user and persistently recorded for auditing and chain-of-custody integrity.
R4.6.3: Audit Logging for Evidence Validation
The system shall log all integrity validation attempts, including:

- User ID performing the validation
- Evidence ID
- Timestamp
- Validation result (`match` / `mismatch`)

R5: Evidence Metadata & Tagging

This section defines how the system handles custom metadata and tags associated with evidence files. Metadata is stored in PostgreSQL during upload, with support for updating, retrieving, and querying by tags for efficient organization and filtering.

R5.1: Metadata Storage (Embedded JSON)

R5.1.1: Embedded Metadata Format
The system shall store metadata associated with an evidence file as a structured JSON object during upload.
R5.1.2: Metata Field Key Management
The system shall enforce validation to ensure metadata keys are non-empty, unique within the object, and do not overlap with reserved system fields.
R5.1.3: Metadata Updates
The system shall allow only system admins to update JSON evidence metadata objects for a given evidence record. This may involve adding new keys(fields), updating values, or removing existing fields.
R5.1.4: Audit Logging for metadata storage and updates
All metadata storage and update actions shall be logged, including:

- User ID
- Evidence ID
- Fields added/updated
- Timestamp

R5.2: Metadata Retrieval (JSONB)

R5.2.1: Metadata Viewing Permissions
The system shall allow users with the `VIEW_EVIDENCE_METADATA` permission to retrieve the `metadata` field of a given evidence record, returned as a JSON object.
R5.2.2: Storage Format
The JSON metadata shall be fully structured and consistent in key-value format.
R5.2.3: Audit Logging for Metadata Retrieval
Each metadata retrieval action shall be logged, including:

- User ID
- Evidence ID
- Timestamp

R5.3: Evidence Tagging

R5.3.1: Users with Tagging Permissions
The system shall allow users who are assigned to a case to tag evidence of that particular case.
R5.3.2: Tag Storage format
Tags shall be normalized (e.g., case-insensitive, trimmed) and stored in a PostgreSQL table with a many-to-many relationship to evidence records.
R5.3.3: Removing Tags
The system shall allow authorized users to remove tags from evidence.
R5.3.4: Audit Logging for Tag operations
The system shall log all tag operations, including:

- User ID
- Evidence ID
- Tag(s) added or removed
- Timestamp

R5.4: Searching and Listing by Tag

R5.4.1: Case Level Evidence Tag Viewing and Filtering
The system shall allow users assigned to a case to list and search for evidence files by tag.
R5.4.2: Searching Feature

Tag-based search shall support exact match, partial match, and compound tag filters.
R5.4.3: Audit Logging for Tag Searching
All search actions involving tags shall be logged with:

- User ID
- Search tags or keywords used
- Timestamp

R6: Audit Logging

This section defines how the system captures and manages logs of critical user actions for accountability, traceability, and compliance. Logs are securely stored and queryable by authorized personnel.

R6.1: Logging of Critical Actions

R6.1.1: Audit Log major system actions
The system shall automatically log all critical user actions, including but not limited to:

- User login and logout
- Evidence upload or deletion
- Case creation and status changes
- Role assignments and revocations
- Metadata updates
- Permission violations or unauthorized attempts

R6.1.2: Audit Log Fields
Each audit entry shall include:

- The action performed (e.g., `LOGIN_SUCCESS`, `UPLOAD_EVIDENCE`, `ASSIGN_ROLE`)
- The user ID performing the action
- The target entity (e.g., case ID, evidence ID, user ID)
- A contextual description (optional, e.g., IP address, browser agent)
- A timestamp (`TIMESTAMPTZ`)

R6.1.3: Audit Log Trigger
Audit logging must be triggered by backend services using a consistent and centralized mechanism (e.g., middleware, service-level logging function).

R6.2: Retrieval and Filtering of Logs

R6.2.1: Recent Activity Viewing
The system shall allow all users to view audit logs of their recent activities.
R6.2.2: Audit Log Retrieval Filtering
The retrieval endpoint shall support filtering by:

- User ID
- Case ID
- Action type
- Time range

R6.2.3: Sorting and Pagination
Results shall be paginated, sorted by newest-first, and returned in a structured format (e.g., JSON array of log entries).

R6.3: Timestamping and Context Preservation

6.3.1: Entry Log Timestamping
Each audit entry shall be stored with a server-generated timestamp reflecting when the action occurred.
R6.3.2: Entry Log Metadata
The system shall capture contextual information if available, including:

- Source IP address
- Request origin (user agent)
- Authenticated session/token ID

R6.3.3: Immutable Log Entries
Audit entries shall be immutable once stored and protected from tampering through access control and restricted write permissions.

R6.4: Exporting Audit Logs

R6.4.1: Exporting Logs
The system shall allow users with the `EXPORT_AUDIT_LOGS` permission to export filtered audit log data in a downloadable format (e.g., CSV or JSON).

R6.4.2: Operations on Logs
The export operation shall apply all currently active filters (user, case, action type, time range) and return only authorized records.
R6.4.3: Log Entries for Export Operations
Each export event shall itself be logged in the audit log, including:

- User ID performing the export
- Export format
- Time of export
- Scope or filters used

R6.4.4: Exported File Details
Exported files shall contain only audit data; no sensitive credentials, secrets, or private keys shall be included.

R7: Dashboard and Analytics

This section outlines requirements for providing users and administrators with summarized, visualized, and actionable insights into system activity, including case trends, evidence flow, and user behavior.

R7.1: Dashboard Overview and Statistics

R7.1.2: User Dashboard Statistics

The dashboard shall display high-level statistics relevant to the user's role and scope, such as:

- Total number of cases accessible to the user
- Number of open vs. closed
- Total number of evidence files uploaded
- Recent user activity (logins, uploads, case status updates)

R7.1.3: Real Time Recent Accesses

Data presented shall reflect real-time or near real-time values from the database, cached appropriately for performance if needed.

R7.2: Charts and Summary Visualizations

R7.2.1: Visual Representation Formats

The system shall provide visual indicators such as:

- Time-series graphs for case creation trends
- Line or area graphs for user activity over time

R7.2.2: Metrics

Summary tiles (KPIs) shall include key metrics such as:

- Number of cases assigned to the current user
- Number of uploads in the last 7 days
- Number of role assignments or changes made

R7.2.3: Visualization Filtering

All visualizations shall support user-level filtering where applicable (e.g., show only my cases, or by role/team).

R7.2.4: Services Provided

The system shall use secure backend APIs to serve aggregated data to frontend dashboards, and ensure no raw sensitive data is exposed unnecessarily.

R8: Notification Service

This section defines requirements for system-generated notifications, primarily focused on account-related email flows such as verification and password reset. Logging is enforced for audit and debugging purposes.

R8.1: Email-Based Notifications

R8.1.1: Email Trigger

The system shall send email notifications for the following events:

- **Email verification** during user registration
- **Password reset requests**, including secure token delivery

R8.1.2: Email Details

Emails shall include:

- A personalized greeting (e.g., user’s full name)
- A secure, time-limited verification or reset link/token
- System branding and support contact information

R8.2: Logging of Email Delivery Attempts

R8.2.1: Audit Logging for Notification

The system shall log every email delivery attempt, including:

- Email type (e.g., `VERIFICATION_EMAIL`, `RESET_PASSWORD_EMAIL`)
- Recipient email address
- Status (`sent`, `failed`, `mocked`)
- Timestamp
- Error message if applicable

R8.2.2: Email Notification Environment

Email logs shall be recorded even in development environments where email delivery is mocked or simulated.

R8.2.3: Notification Logs Viewing

Logs shall be accessible to administrators or developers with the `VIEW_NOTIFICATION_LOGS` permission for debugging and auditing.

R9: Secure File Access (E2EE-Ready)

This section defines how the system provides encrypted file access using IPFS and end-to-end encryption mechanisms. It ensures that only authorized recipients, through a secure key exchange protocol, can decrypt and access evidence.

R9.1: Downloading Encrypted Files from IPFS

R9.1.1: IPFS File Download Using CID
The system shall allow users who are assigned to a case to initiate a file download by referencing the files IPFS CID.

R9.1.2: End to End Encryption
All files stored on IPFS shall be encrypted client-side before upload, and decrypted client-side after download using the appropriate decryption key.

R9.1.3: Storage of Encrypted Artefacts
The backend shall **never store or access plaintext file contents** — it shall only relay encrypted blobs and metadata.

R9.2: Key Exchange Protocol Integration (X3DH)

R9.2.1: Key Exchange Protocol
The system shall integrate with a secure key exchange protocol such as **X3DH (Extended Triple Diffie-Hellman)** to enable encrypted key sharing between sender and recipient(s).

R9.2.2: Encryption Keys
The sender shall encrypt a file encryption key using the recipient's X3DH public keys and attach the encrypted symmetric key as part of the evidence metadata or envelope.

R9.2.3: Decryption Keys
The recipient shall use their identity and ephemeral keys to decrypt the symmetric file key and then decrypt the actual file locally.

R9.2.4: Retrieval Processes
All key exchange and retrieval processes shall occur over authenticated, secure channels.

R9.3: Authorization and Decryption Access Control

R9.3.1: RBAC Enforced on File Sharing
The system shall ensure that only users authorized via RBAC and case-specific roles can retrieve the encrypted file or associated key material.

R9.3.2: Authorization and Authentication of Shared Files
Upon download request, the system shall verify:

- The requesting user's identity and role
- That the user is assigned to the case associated with the evidence
- That the user is the intended recipient of the encrypted key (via public key match)

R9.3.3: Audit Logging Upon Unauthorized Access
Access attempts by unauthorized users shall be denied and logged with an audit entry indicating attempted access to secured content.

R10: Case Collaboration

This section defines functionality for users to collaborate within a case using notes, replies, and tagging. Notes are role-protected, timestamped, and audit-logged to ensure accountability and traceability.

R10.1: Posting Case Notes

R10.1.1: Case-Level Note Sharing
The system shall allow users who are assigned to a case to post written notes to a specific case.

R10.1.2: Format of Notes
Notes may contain plain text, markdown, or basic formatting for clarity and context.

R10.2: Replying to Case Notes

R10.2.1: Discussion Threads
Users shall be able to reply to existing notes, forming threaded discussions within the same case.

R10.2.2: Note Ordering
Replies shall be visually grouped with their parent notes and ordered chronologically.

R10.3: Timestamping and Author Tracking

R10.3.1: Audit Logging for Case Notes
Every case note and reply shall be stored with:

- Author’s user ID
- Timestamp of creation
- Case ID
- Parent note ID (for replies)

R10.4: Viewing Case Discussions

R10.4.1: Case-Level Note Viewing
The system shall allow users assigned to a case to view all case notes they are authorized to access, presented in a chronological or threaded format.

R10.4.2: Case-Level Note Grouping
Notes shall be grouped per case and loaded with associated metadata (e.g., author name, created_at).

R10.5: User Tagging for Directed Collaboration

R10.5.1: Tagging Users Within a Case Note
The system shall support tagging users within case notes using a standard `@username` format.

R10.5.2: Tagging for System Notification
When tagged, the mentioned user shall receive a system notification or email (if notification service is active).

R10.5.3: Tagging Range
The system shall validate that only users assigned to the same case may be tagged.

R10.6: Access Control

R10.6.1: Case-Level Note Access Control
Only users who are assigned to the case shall be allowed to create, view, or reply to notes within that case.

R10.7: Auditing Note Activity

R10.7.1: Audit Log All Note Replies
The system shall log all note and reply actions in the audit log, including:

- Action type (`add_note`, `reply_note`)
- Author’s user ID
- Case ID
- Note or reply ID
- Timestamp

R11: Secure Chat (Real-Time Messaging)

This section defines how the system supports secure, encrypted real-time messaging using authenticated WebSockets. It enables one-on-one and group conversations within cases while enforcing strict encryption and access control.

R11.1: WebSocket Authentication

R11.1.1: JWT/Session Token Authentication
The system shall establish WebSocket connections only after authenticating the user using a valid **JWT** or **session token**.

R11.1.2: Authorization Upon Connection
The WebSocket server shall verify the token on connection and reject unauthorized or expired sessions.

R11.1.3: Websocket Linking
Each connected socket shall be linked to the authenticated user ID and limited to their assigned case channels or private message threads.

R11.2: Encrypted Messaging

R11.2.1: End-to-End Messaging
All messages shall be end-to-end encrypted **client-side before transmission**, using pre-shared symmetric keys or session-derived keys.

R11.2.2: Single And Grouped Encryption
The system shall support both:

- **One-on-one encrypted messages** between two users

- **Group encrypted messages** within a case-specific chat

R11.2.3: Messaging Keys and Encryption Protocol
Message encryption keys shall be managed using a secure key exchange protocol (e.g., X3DH or Double Ratchet).

R11.3: Real-Time Delivery

R11.3.1: Real Time Messaging
The system shall deliver messages in real-time using a WebSocket-based gateway service.
R11.3.2: Access Control for Delivered Messages
The gateway shall route messages only to authenticated clients who are valid recipients.
R11.3.3: Undelivered Messages
Undelivered messages shall be queued or temporarily stored for users who are offline.

R11.4: Encrypted Message Storage

R11.4.1: Encrypted Messages
The system shall persist encrypted message payloads in the database, along with:

- Sender ID
- Recipient ID(s) or case ID
- Message timestamp
- Delivery status (`sent`, `delivered`, etc.)

R11.4.2: Storage of Encrypted Messages
Plaintext content shall never be stored server-side.

R11.5: Delivery Acknowledgment

R11.5.1: Message Acknowledgement Status Update
The system shall support acknowledgment of message delivery and update the message status to:

- `sent` (message was dispatched)
- `delivered` (message reached the recipient's client)

R11.6: Message Visibility & Access Control

R11.6.1: Viewing Case-Level Messages
Only users assigned to the same case (or valid peer in one-on-one chat) shall be able to send or view messages related to that conversation.
R11.6.2: Authorization of Case-Level Messages
Attempts to access messages outside of one’s assigned cases or private

conversations shall be blocked and logged.

R11.7: Message Retrieval (Encrypted History)

R11.7.1: Retrieval of Encrypted Messages
The system shall allow authenticated users to retrieve their encrypted message history from the backend.
R11.7.2: Client Side Decryption
The backend shall return only encrypted payloads — decryption shall occur on the client using previously negotiated or stored keys.
R11.7.3: History Retrieval
History retrieval shall support pagination and filtering by time or conversation.

R11.8: Encryption Key Management

R11.8.1: Key Rotation
The system shall support secure key rotation or refresh per session, per case, or per user — based on defined security policies.
R11.8.2: Invalidation of Keys
Old keys may be invalidated or archived securely. Key refresh shall be securely negotiated and authenticated using the key exchange protocol.
R11.8.3: Key Storage
Key material shall **never be stored unencrypted** on the backend. All key storage (if any) must be encrypted using a derived key from the user's credentials or device-based secrets.

R12: End-to-End Encryption (E2EE) for Chat

This section outlines how the system uses strong cryptographic protocols to ensure chat messages remain confidential, authentic, and tamper-proof, even from the server. The design follows modern E2EE standards like **X3DH** and optionally **Double Ratchet**.

R12.1: Key Agreement with X3DH

R12.1.1: Encryption Protocol
The system shall use the **X3DH (Extended Triple Diffie-Hellman)** protocol to establish secure shared secrets between sender and recipient.
R12.1.2: Encryption Keys
The following are published on behalf of a user by the system:

- An **Identity Key (IK)**
- One or more **Signed PreKeys (SPK)**
- A set of **One-Time PreKeys (OTPKs)**

R12.1.3: Retrieval of Keys Upon Connection
Upon initiating a new session, the sender shall fetch the recipient's pre-key bundle and derive a shared encryption key using X3DH.

R12.2: Secure Key Storage

R12.2.1: Secure Key Storage
The system shall store users' Identity Keys and PreKeys in a secure **Key Service** or backend module with:

- Encrypted storage (e.g., libsodium's `crypto_secretbox`)
- Access control restricted to the authenticated owner
- Optional client-side derived encryption for Identity Keys using password-derived keys

R12.2.2: Sharing of Only Public Keys
Only the public parts of the keys (IK, SPK, OTPKs) shall be accessible to other users for key exchange. Private keys shall never leave the user's device unencrypted.

R12.3: Client-Side Message Encryption

R12.3.1: Client Side Encryption
All chat messages shall be encrypted **on the sender's device** using a session key derived from X3DH (and optionally advanced to support the Double Ratchet protocol).

R12.3.2: Metadata Storage
Encrypted payloads shall be accompanied by a metadata envelope that includes:

- Sender public key
- Ephemeral public key (if using Double Ratchet)
- Message counter (if using forward secrecy)
- Message authentication code (MAC) or signature

R12.4: Client-Side Message Decryption

R12.4.1: Decryption Keys
Only the **intended recipient** shall be able to decrypt the message using their private identity key and the sender's ephemeral/public keys.

R12.4.2: Client-side Decryption
Message decryption shall occur entirely on the recipient's device — no server-side decryption is allowed.

R12.4.3: Chat Payload
In one-to-one chats, each recipient gets a uniquely encrypted payload. In group chats, per-user encryption or sender keys shall be applied depending on the model.

R12.5: Server Transparency & Content Protection

R12.5.1: No Server Side Decryption
The server shall act solely as a **message relay** and shall be **unable to decrypt, alter, or inspect** message contents.

R12.5.2: Messaging Format
All stored chat messages (in transit and at rest) shall remain in their encrypted form. The system shall not log or index plaintext chat content.

R12.6: Forward Secrecy and Session Expiry

R12.6.1: Forward Secrecy
The system shall support **forward secrecy** by rotating session keys periodically or after each message, using the **Double Ratchet** protocol or equivalent.

R12.6.2: Session State
Session state shall expire after:

- A period of inactivity
- Session key compromise
- A user logout or explicit key reset

R12.6.3: Re-establishing a Session
Upon session expiration, a new shared secret must be re-established via X3DH before communication can continue.

R12.6.3: Terminating a Session
The system shall terminate a session after an extensive period of inactivity is detected.

R13: Case Timeline Management

R13.1: Event Entry and Linking
The system shall support the creation, linking and reordering of events

R13.1.1: Event Creation
The system shall allow DFIR team members to create events in a chronological timeline. Each event shall include:

- Date and time
- Event description
- Linked evidence items

R13.1.2: Linking Evidence to Events
The system shall allow each timeline event to be linked to one or more evidence items. Linked evidence shall reference the evidence ID and include a short description of relevance.

R13.2: Timeline Editing and Review

R13.2.1: Event Editing
Lead DFIR Investigators shall be able to edit, remove, or reorder timeline events for accuracy and completeness.

R13.2.2: Event Approval
Investigators shall be able to approve events, confirming that linked evidence and descriptions are accurate.

R13.3: Timeline Navigation and Search
The system shall provide filtering and search capabilities to view events by:

- Date or date range
- Event type
- Linked evidence

R13.4: Session and Security Considerations
Timeline modifications shall be logged automatically, including actor ID, timestamp, and action type. Only authorized users with access to the case shall modify or view timeline events.

R14: Graphical Evidence Mapping

R14.1: Evidence Node Management
R14.1.1: Node Creation
The system shall allow DFIR team members to create nodes representing evidence items. Each node shall include:

- Evidence ID
- Evidence type
- Optional short description

R14.1.2: Node Linking
The system shall allow users to visually link nodes to represent relationships, dependencies, or causal connections between evidence items.

R14.2: Map Editing and Highlighting
R14.2.1: Map Editing
Lead DFIR Investigators shall be able to add, edit, or remove nodes and links to ensure accuracy and highlight critical relationships.

R14.2.2: Node/Link Annotations
Users shall be able to annotate nodes or links with notes, comments, or references to timeline events. External Collaborators shall have annotations clearly marked as “External Collaborator.”

R14.3: Map Viewing and Export
The system shall allow authorized users to:

- View the full graphical evidence map
- Export the map in visual formats (PDF, PNG) for reporting or external review

R14.4: Security and Access Control
All map modifications shall be logged automatically. Only authorized users with case

access shall view or edit maps. Read-only access shall be enforced for external collaborators unless temporary edit access is explicitly granted.

R15: Case Report Generation

R15.1: Report Composition

R15.1.1: Report Content

The system shall generate structured case reports that may include:

- Evidence items and metadata
- Annotation threads
- Timeline events
- Graphical evidence maps
- Executive summary or custom notes

R15.1.2: Section Selection

Lead DFIR Investigators shall be able to include or exclude specific sections when generating reports.

R15.2: Report Export and Formats

R15.2.1: Export Options

Reports shall be exportable in multiple formats, including PDF and DOCX.

R15.2.2: Preview

The system shall allow users to preview reports prior to export to verify content and layout.

R15.3: Security and Audit

R15.3.1: Access Control

Only authorized users shall generate or export reports. External collaborators may view reports only if explicitly granted access.

R15.3.2: Logging

All report generation actions shall be logged automatically, including user ID, timestamp, and report version.

ARCHITECTURAL & QUALITY REQUIREMENTS

This section outlines the system's architectural principles and non-functional (quality) requirements, including modularity, scalability, performance, security, and reliability. These requirements guide technical decision-making and ensure the platform supports robust, maintainable, and legally defensible DFIR workflows.

Quality Requirements

Performance Requirements

Performance is an important quality attribute for AEGIS. The system’s primary purpose is to provide a platform that accommodates DFIR teams simultaneous collaboration during ongoing investigations. Thus, reasonable response times are key to maintain workflow continuity without affecting the user's analysis process.

Stimulus Source	Platform user
Stimulus	Users accessing case management features, uploading evidence files, and engaging in real-time collaboration during active investigations
Response	● The system provides immediate feedback for user interactions ● Evidence upload and case access operations complete without delays ● Real-time collaboration features remain responsive during simultaneous use
Response Measure	● Web interface pages load within **2 seconds** ● Evidence files up to **1GB** upload within **10 seconds** ● Real-time collaboration features respond within **500 milliseconds** ● The system handles **5 concurrent users** without performance degradation
Environment	Normal operational conditions
Artefact	AEGIS web interface, database queries, evidence upload system, and real-time collaboration features

Performance Strategies

Caching: Store frequently accessed data in memory to reduce database queries and improve response times for common operations.

AEGIS will cache case metadata, user permissions, and frequently accessed evidence listings in memory within the Go backend to avoid repeated database queries, particularly for cases accessed multiple times during active investigations.

Indexing: Create database indexes on frequently queried fields to improve search performance and reduce query response times.

AEGIS will index evidence metadata fields (case ID, file hash, timestamps) in the database to ensure evidence searches and case lookups complete within the 1-second target response time.

Performance Patterns

- Service Mesh
- Load Balancer
- Throttling
- MapReduce

Scalability Requirements

DFIR teams can consist of a variable number of members, resulting in highly variable volumes of case data and user activity. AEGIS must be capable of scaling dynamically to accommodate both small teams handling isolated incidents and large organizations managing multiple complex investigations in parallel. This includes supporting high-throughput evidence ingestion, concurrent annotation and messaging workflows, and efficient retrieval of case artefacts.

Stimulus Source
Multiple investigators working on cases simultaneously
Stimulus
Increasing number of concurrent users accessing cases, uploading evidence, and using collaboration features
Response
The system maintains functionality as user load increases Case and evidence data can grow without system failure

Collaboration features work consistently across multiple users
Response Measure
The system supports **5 concurrent users** without failure Handles **20 total evidence files** across all test cases Real-time collaboration supports **3 simultaneous annotation threads** The database maintains response times under **2 seconds** with **100 total evidence records**
Environment
Design of the system and deployment strategy
Artefact
Authentication system, case management features, evidence storage, and real-time collaboration components

Scalability Strategies

Scale-up resources (increase processing power): Add more computing capacity to handle increased workload and user demands by upgrading hardware components or utilizing containerized scaling.

For AEGIS, we will use Docker containers that can be scaled vertically by increasing CPU and memory allocation when concurrent user loads exceed thresholds, particularly during large incident response operations.

Spread load across time: Distribute workload evenly by implementing queuing mechanisms for resource-intensive operations to prevent system overload during peak usage.

When multiple large evidence files are uploaded simultaneously, concurrent users perform complex searches, or multiple chat conversations occur, AEGIS will queue these operations and manage chat message delivery efficiently to prevent system bottlenecks.

Scalability Patterns

- Microservices

- Service-oriented Architecture (SOA)

Security

Security is a critical quality attribute in the AEGIS platform, as it must protect digital evidence from unauthorized access, alteration, or deletion. There is a need to prevent evidence tampering through encryption, role-based access control (RBAC), and exhaustive logging to maintain evidence integrity and confidentiality. Given the sensitivity of digital evidence, especially in legal or criminal investigations, unauthorized access or changes could compromise the entire case. Malicious insiders or external attackers may attempt to breach the system; hence, enforcing security through multiple defense layers is essential. Security mechanisms such as end-to-end encryption, logging of all access attempts, and enforcing strict RBAC ensure only authorized users can access or manipulate the data, while all actions are auditable. By implementing this level of security, AEGIS fulfils the goal of ensuring evidence integrity.

Stimulus Source
External attacker or malicious insider
Stimulus
Attempts to access, alter, or delete stored evidence
Response
System denies unauthorised access, encrypts and logs all interactions
Response Measure
· All unauthorised access is blocked · All accesses logged · Data at rest and in transit is encrypted · RBAC rules are enforced per role
Environment
During normal operation and when evidence is being accessed or uploaded
Artefact
Evidence files and logs

Architectural strategy(s):

Authentication:

Authentication ensures only verified users can interact with the platform. AEGIS requires role-based access and user-specific permissions to limit access to sensitive evidence. Without strong authentication, unauthorised users may bypass controls. Implementing secure authentication guarantees that only legitimate users enter the system. This reinforces the security posture of the system and complements RBAC, a critical requirement for digital forensics platforms.

Authorisation:

Authorisation restricts what authenticated users can access or do. The system implements RBAC to limit visibility and access based on assigned roles. Even among trusted users, not all should access all evidence. For instance, a DFIR team member shouldn't have admin privileges. RBAC ensures fine-grained control over actions and data visibility. This aligns with legal compliance requirements for access control and minimises insider threat vectors.

Audit logging:

Audit logs provide a traceable record of all user and system actions. The platform logs every activity for audit and chain-of-custody purposes. Logs ensure that any action, authorized or attempted unauthorized, is recorded for forensic reconstruction, debugging, or legal verification. Logging is essential to maintain chain-of-custody, detect breaches, and support accountability in a legal context.

Hashing and integrity checks:

Hashing validates that data has not been tampered with. While not explicitly named, the proposal's emphasis on integrity suggests the use of hashing for verifying evidence authenticity. When evidence is uploaded or accessed, hashing (e.g., SHA-256) ensures no bits have changed. This is critical for legal admissibility of digital evidence. Supports the core aim of preventing tampering and enabling provable evidence integrity

End-to-end encryption:

Encryption protects evidence confidentiality during transit and at rest.AEGIS mandates both end-to-end encryption for communication and encryption at rest for stored evidence. Encryption ensures that even if data is intercepted or exfiltrated, it remains unreadable to unauthorized actors. It protects against external breaches and internal data leaks. Fulfills compliance requirements and aligns with best practices for safeguarding forensic data.

Architectural pattern:

Microkernel

This pattern promotes modular security services that can be independently updated.

Availability

Availability is a vital quality attribute in the AEGIS platform, as uninterrupted access to evidence and services is critical during time-sensitive investigations. The system must have high uptime, rapid failover, and minimal recovery time while ensuring no data loss or corruption during failures. In forensic operations, even short outages can disrupt workflows, delay evidence analysis, and jeopardize legal procedures. High availability ensures that parts of the system remain functional during compartmental failures. This includes real-time collaboration, uninterrupted access to evidence files, and responsive communication features. Automated recovery mechanisms are essential to mitigate service disruptions. Ensuring availability directly supports the platform's mission, which is to facilitate efficient, collaborative forensic investigations.

Stimulus Source
User (investigator), automated monitoring service, or another subsystem
Stimulus
A request is made to access the system or evidence while a component is down
Response
The system reroutes the request, activates a backup service, or degrades gracefully
Response Measure
· System uptime > 99.3% · Has a Mean Time To Repair (MTTR) of at most 2 hours. · Automatic failover in <5 seconds · No data loss ot corruption
Environment
During normal operation or when the system is under heavy load
Artefact
Evidence storage, collaboration module (i.e., secure chat, annotation threads), communication service

Architectural strategy(s):

Health monitoring:

Health monitoring continuously tracks system performance and detects failures early.The system is expected to initiate failover or restart components when a service becomes unavailable. This allows rapid detection and recovery, supporting the MTTR and failover targets, thus maintaining high availability.

Automated restarts:

Automated restarts allow failed components to be rebooted without manual intervention. The platform requires minimal downtime and quick recovery, with MTTR < 2. Restarting containers, services, or nodes automatically ensures service continuity without waiting for human operators. This is particularly useful during temporary crashes or memory leaks.

Redundancy:

Redundancy ensures that a backup instance is always available to take over if the primary one fails. The system must have automatic failover in under 5 seconds and ensure no data loss. Redundancy directly ensures uptime and zero data loss, aligning with the platform's 99.5% availability requirement.

Architectural pattern(s):

Microservices

This pattern maximises availability by ensuring separation of concern and support for redundancy and failover mechanisms. The client initiates the request, and the server can replicate that request in high-availability clusters.

Reliability

Reliability is critical for AEGIS because digital forensic investigations require the system to operate consistently and without failure, preserving evidence integrity and ensuring uninterrupted workflow. Forensic systems like AEGIS handle time-sensitive and legally sensitive tasks. If the system corrupts evidence, fails mid-operation, or cannot quickly recover from faults (like power surges or failed uploads), it can compromise the entire investigation. By ensuring reliable operation with data validation, auto-recovery, and fault isolation, the platform can maintain investigator trust and deliver consistent performance, even under stress or partial system failure.

Stimulus Source
Power surges, corrupted backups, transmission errors, internal component failures
Stimulus
A failure occurs during evidence upload, access, storage, or collaboration

Response
The system prevents data corruption, recovers from failure, and resumes operation
Response Measure
· No loss of uploaded evidence · Auto-recovery time < 10 seconds · Error logged · Has a mean time between failures of at least 1 week
Environment
While the system is actively being used in an ongoing investigation
Artefact
Evidence database, file storage system, logging mechanism

Architectural Strategy(s):

Fault detection and isolation:

Fault detection and isolation allow the system to identify and isolate faulty components without affecting the whole system. If one component (e.g., the chat module) fails, isolation ensures the failure doesn't crash the entire platform. Health checks and logging tools can detect errors, while self-healing routines or service restarts can isolate and fix the fault.

Data validation and integrity checks:

These techniques ensure data is correct and unaltered during upload, storage, and access. AEGIS is required to prevent data corruption and maintain evidence integrity even during transmission errors or component failures. Validation (e.g., file type checks, hashing) detects anomalies, and integrity checks (e.g., checksums, signatures) ensure data hasn’t been modified or corrupted. In turn, reliable evidence handling, a cornerstone requirement in forensic systems, is accomplished.

Backup systems and transactional operations:

Backups and transactions allow the system to recover from failure points and avoid partial or inconsistent states. The system must avoid any data loss or corruption, even during power surges or interrupted uploads. Backups capture the system state periodically. Transactional operations (e.g., ACID-compliant database operations) ensure that incomplete actions don’t corrupt the system. On failure, the system can roll back or restore a previous valid state.

Architectural pattern:

Repository:

The repository pattern centralises access to data and manages interactions with the underlying storage. The reliability target involves no evidence loss and consistent data handling across the file storage and logging systems. Repositories abstract data access and can include validation, transaction control, and failover support. This makes it easier to enforce consistency and data recovery logic. This ensures consistent, reliable access to and modification of data, which is foundational for forensic reliability.

Architectural Constraints

Architectural constraints define non-negotiable design boundaries that the AEGIS system must respect due to compliance, technical environment, and project limitations.

1. The system must handle sensitive forensic evidence, so it must meet legal and regulatory compliance.
    a. There is a compliance constraint to data privacy laws, i.e., POPIA, which may prohibit storing sensitive evidence off-site, pushing for on-premises hosting.

2. The chain of custody must be enforced, requiring tamper-proof logs and documented data handling.
    a. Chain-of-custody enforcement demands immutability and full traceability. You can't just overwrite data or delete logs.

3. It must run in an on-site deployment environment, not the public cloud.

4. The system uses PostgreSQL, MongoDB, IPFS, and Docker for all components.
    a. IPFS doesn't support standard database features like transactions. This means we can’t rely on it to safely undo or roll back operations. So, we must make sure that actions like uploading evidence are repeatable without causing problems, and we must check that files haven't been changed, using things like hashes.

    b. Since we’re using Docker to run everything in containers, we have to choose tools that work well in containers. Docker also means we need to plan carefully for how services will talk to each other, how they will be scaled up or down, and how we’ll restart them automatically if something fails.

TECHNOLOGY CHOICES AND REQUIREMENTS

This section justifies the selected technologies for implementing the AEGIS platform, including backend frameworks (Go), frontend frameworks (React with Tailwind), databases (PostgreSQL), messaging systems (NATS/RabbitMQ), and containerization tools (Docker, Kubernetes). Each choice is aligned with project goals such as scalability, security, and developer productivity.

Case Timeline/Audit Log Viewer

Tech Area: Database

- **Options Considered:** PostgreSQL, MongoDB, Elasticsearch
- **Final Choice:** MongoDB
- **Justification:**
    - MongoDB stores data in flexible JSON-like documents, making it ideal for storing logs with dynamic and evolving structures.
    - Offers high write performance, which is critical when handling large volumes of audit or timeline events.
    - Eliminates the need for predefined schemas, allowing different types of events to be logged without structural constraints.
    - Performs well for read-heavy filtering queries, especially when properly indexed.

Tech Area: Log Storage Format

- **Options Considered:** Flat text logs, Structured JSON, Log files
- **Final Choice:** Structured JSON
- **Justification:**
    - Structured JSON enables efficient filtering and querying, such as by user, timestamp, or action.
    - Better integration with frontend filters and visualizations than plain text or log file formats.

    - It supports the storage of nested data and metadata (e.g., IP, case ID, user agent), which is often useful in security and audit contexts.

Tech Area: Frontend Framework

- **Options Considered:** React, Angular, Vue
- **Final Choice:** React
- **Justification:**
    - Ensures consistency across modules.
    - React's component-driven architecture is well-suited for dynamic and interactive UIs like timeline viewers.
    - Large ecosystem of supporting libraries (e.g., for date filtering, animations, and charts).
    - Simplifies integration of stateful features such as search, filter, zoom, or hover-over metadata.

Tech Area: Timeline UI Library

- **Options Considered:** Custom UI, Vis.js Timeline, React Flow Timeline
- **Final Choice:** Custom React Component
- **Justification:**
    - Full control over design, styling, and interactivity, tailored to the visual identity and UX patterns.
    - Avoids dependency on heavy or unmaintained third-party libraries, reducing bundle size and future tech debt.
    - Enables fine-grained customization (e.g., colored event markers by type, expandable entries, grouped timelines).
    - Allows easy extension for hover details, click-to-expand, or linked evidence previews.

Tech Area: API Layer (Backend)

- **Options Considered:** Go REST API, GraphQL, WebSocket
- **Final Choice:** Go REST API
- **Justification:**
    - REST is lightweight, stateless, and well-supported across clients.
    - Facilitates clear, filterable endpoints for frontend integration.
    - Easier to cache, paginate, and audit compared to GraphQL or streaming approaches.

Tech Area: Timestamp Precision

- **Options Considered:** UNIX timestamp, ISO 8601 string
- **Final Choice:** ISO 8601
- **Justification:**
    - Human-readable format, widely understood across APIs and user interfaces.
    - Compatible with frontend date/time pickers and timeline displays without extra formatting.
    - Retains time zone awareness and improves debugging and analysis by humans.

Tech Area: Authentication

- **Options Considered:** JWT, Sessions, OAuth2
- **Final Choice:** JWT (JSON Web Tokens)
- **Justification:**
    - Stateless design simplifies backend logic and scales well across distributed systems.
    - Embeds user identity, roles, and claims directly into the token—this makes user attribution of log entries straightforward.

    - Ensure seamless integration and consistency**.**
    - Supports expiration, signature verification, and secure transmission.

Tech Area: Log Storage Database

(repetition of Database decision, but focused on logs)

- **Options Considered:** MongoDB, PostgreSQL, Elasticsearch
- **Final Choice:** MongoDB
- **Justification:**
    - Optimal for storing structured logs as JSON documents.
    - Supports flexible and nested fields, perfect for variable log types.
    - Scales well with high log ingestion rates.
    - Easy to index and filter logs for building performant timelines or audit reports.

Evidence Upload, Storage & Retrieval

Tech Area: Storage Backend

- **Options Considered:** Local File System (FS), AWS S3, ownCloud (WebDAV), IPFS
- **Final Choice:** IPFS (InterPlanetary File System)
- **Justification:**
    - Decentralised and content-addressable, meaning files are stored by hash rather than location.
    - Guarantees tamper-evidence: any change to the file alters its hash.
    - Perfect for chain-of-custody use cases, ensuring traceability and immutability.
    - Resilient and distributed storage, not tied to a single server or vendor.

Tech Area: Hashing Algorithm

- **Options Considered:** MD5, SHA-1, SHA-256
- **Final Choice:** SHA-256
- **Justification:**
  - cryptographically safe, in contrast to SHA-1 and MD5, which are known to have flaws.
  - Widely used in security and forensic contexts for integrity verification.
  - Compatible with most modern libraries, standards, and chain-of-custody systems.

Tech Area: Upload Format

- **Options Considered:** Multipart form-data, base64 in JSON, file streaming
- **Final Choice:** Base64 in JSON
- **Justification:**
  - Easy to embed in JSON-based REST APIs, which simplifies backend parsing.
  - Plays well with end-to-end encryption on the client side before upload.
  - Reduces reliance on multipart parsing and simplifies debugging in tools like Postman.

Tech Area: Backend Language

- **Options Considered:** Node.js, Go, Python
- **Final Choice:** Go
- **Justification:**
  - High-performance language with built-in concurrency, ideal for handling multiple large uploads in parallel.
  - Produces lightweight binaries and is resource-efficient, improving scalability.

Tech Area: API Design

- **Options Considered:** REST API, GraphQL, gRPC
- **Final Choice:** REST API
- **Justification:**
    - REST is easy to integrate, test, and monitor using standard tools.
    - Matches the API design already in use across the AEGIS platform.
    - Provides straightforward routes for file operations.

Tech Area: Metadata Storage

- **Options Considered:** PostgreSQL, MongoDB, Hybrid
- **Final Choice:** PostgreSQL + MongoDB (Hybrid)
- **Justification:**
    - PostgreSQL stores structured relationships: evidence IDs, case associations, timestamps, and uploader info.
    - MongoDB stores flexible/unstructured data: user annotations, evidence tags, and extracted metadata.
    - This hybrid model offers strong consistency for core data while allowing schema flexibility where needed.

Tech Area: Encryption

- **Options Considered:** At-rest only, Client-side E2EE, Server-side encryption
- **Final Choice:** Client-side End-to-End Encryption
- **Justification:**
    - Files are encrypted before leaving the browser, supporting a zero-trust architecture.

    - Backend and storage systems never see plaintext—enhancing privacy and reducing liability.
    - Aligns with modern security best practices for sensitive evidence handling.

Tech Area: Upload Verification

- **Options Considered:** None, checksum revalidation, automatic integrity checks
- **Final Choice:** Hash + Size Checks
- **Justification:**
    - Validates the file hash (SHA-256) and size during both upload and retrieval to detect tampering.
    - Helps ensure data integrity at every stage of the upload lifecycle.
    - Makes the system forensically sound and legally defensible.

Case Creation, Assignment, and Tracking

Tech Area: Database

- **Options Considered:** MySQL, PostgreSQL, MongoDB
- **Final Choice:** PostgreSQL
- **Justification:**
    - Offers robust relational modelling, perfect for structured data like cases, users, and roles.
    - Supports foreign keys, constraints, and auditing features.
    - Reliable and battle-tested for transactional integrity and consistency.

Tech Area: Case Status + Stages

- **Options Considered:** Free-form strings, ENUM types, lookup tables
- **Final Choice:** PostgreSQL ENUMs
- **Justification:**
    - Provides type-safe status definitions, reducing errors during updates.
    - Enforces consistency across cases for workflow stages.
    - ENUMs are performant and easy to validate at the DB level.

Timeline Subsystem (Case Timeline)

Tech Area: Timeline Visualization

- **Options Considered:** Custom React component, Vis.js Timeline, React Flow, D3.js
- **Final Choice:** Custom React component
- **Justification:**
    - Tailored UX for AEGIS; supports color-coded events, grouping, and hover metadata.
    - Avoids dependency bloat or outdated third-party libraries.
    - Supports easy integration with filters, search, zoom, and click-to-expand interactions.

Tech Area: Event Filtering & Sorting

- **Options Considered:** Backend filtering only, Frontend filtering, Hybrid approach
- **Final Choice:** Hybrid (backend + frontend)
- **Justification:**
    - Backend ensures only relevant events are sent, improving performance.
    - Frontend provides instant interactivity for sorting/filtering without repeated API calls.

Tech Area: Event Linking

- **Options Considered:** Simple click-to-expand, Click-to-evidence, Hover popovers
- **Final Choice:** Click-to-expand + Hover metadata
- **Justification:**
    - Improves usability for investigators.
    - Hover provides quick context; click allows deeper navigation into related evidence or cases.

Tech Area: Pagination / Lazy Loading

- **Options Considered:** Load all events, Infinite scroll, Paginated requests
- **Final Choice:** Paginated requests / infinite scroll hybrid
- **Justification:**
    - Reduces memory footprint on the frontend.
    - Ensures a smooth timeline experience for cases with thousands of events.

Graphical Mapping Subsystem (IOC / Evidence Graphs)

Tech Area: Graph Rendering

- **Options Considered:** D3.js, Cytoscape.js, Vis.js Network, React Flow
- **Final Choice:** Cytoscape.js with React wrapper
- **Justification:**
    - Powerful graph layout algorithms, clustering, and zoom/pan.
    - Easy integration with React components and existing AEGIS theming.
    - Supports interactive node expansion, edge highlighting, and metadata popovers.

Tech Area: Interactivity / Node Metadata

- **Options Considered:** Hover only, Click only, Hover + Click
- **Final Choice:** Hover + Click
- **Justification:**
  - Quick inspection via hover.
  - Deeper investigation via click (e.g., opens evidence detail or case summary).

Tech Area: Graph Storage / Update

- **Options Considered:** Precomputed graphs (batch), Dynamic queries (real-time)
- **Final Choice:** Dynamic queries with caching
- **Justification:**
  - Keeps graph current with new evidence or relationships.
  - Caching improves performance for commonly accessed subgraphs.

SERVICE CONTRACTS

This section defines the input/output contracts for each service in the AEGIS platform. It formalizes API interfaces, request/response structures, preconditions, and postconditions for all key operations, supporting consistent backend development and enabling effective testing and integration.

i Swagger Editor has evolved! Codegen, save, and export features are now part of Swagger Pro. Learn More Try Swagger Pro

Sign in

File ▾ Edit ▾ Generate Server ▾ Generate Client ▾ About ▾ Try to API Hub API Hub

Errors Show

Authorize

AEGIS Platform API

Cases

1.0 OAS 2.0

[Base URL: localhost:8080/api/v1]

API for collaborative digital forensics investigations.

Contact AEGIS Support

Apache 2.0

GET `/api/v1/cases/{case_id}/collaborators` Get case collaborators

Retrieves all collaborators (users with roles) for a specific case. Requires authentication.

Parameters Try it out Name Description

Name	Description
case_id *(path)* required * `string`	case_id Case ID

Responses **Response content type** **application/json**

Code **Description**

200 Collaborators retrieved successfully

Example Value Model

```
{
  "data": [
  {
  "assigned_at": "string",
  "email": "string",
  "full_name": "string",
  "id": "string",
  "role": "string"
  }
  ]
}
```

400 Invalid case ID

Example Value Model

```
{
  "details": "string",
  "error": "string",
  "message": "string"
}
```

401 Unauthorized

Example Value Model

```
{
  "details": "string",
  "error": "string",
  "message": "string"
}
```

500 Internal server error

Model
Example

Value {

```
  "details": "string",
 "error": "string",
 "message": "string" }
```

POST `/api/v1/cases/{case_id}/collaborators` Assign user to case
Assigns a user to a case with a specific role. Only users with 'Admin' role can perform this action.

Parameters Try it out Name Description

case_id * required

`string`

`(path)`

Case ID

case_id

request * required

Assignment

Request

`object`

`(body)`

Example

Value {

Model

```
  "role": "string",
  "user_id": "string"
 }
```

Parameter content type

application/json

Responses Response content type application/json

Code Description

201 User assigned successfully

Value
Example

```
"string"
```

Model

400 Invalid request payload

Model

Example

Value

```
{
  "details": "string",
  "error": "string",
  "message": "string"
}
```

401 Unauthorized - Authentication required

Value

Example Model

Code **Description**

```
{
  "details": "string",
  "error": "string",
  "message": "string"
}
```

403 Forbidden - Admin privileges required

Model

Example

Value

```
{
  "details": "string",
  "error": "string",
  "message": "string"
}
```

500 Internal server error

Model

Example

Value

```
{
  "details": "string",
 "error": "string",
 "message": "string"
}
```

DELETE `/api/v1/cases/{case_id}/collaborators/{user_id}` Unassign user from case

Removes a user from a case. Only users with 'Admin' role can perform this action.

Parameters Try it out

Name	Description
case_id * required `string` *(path)*	Case ID case_id
user_id * required `string` *(path)*	User ID user_id

Responses

Response content type application/json

Code	Description
200	User successfully removed from case **Example Value** Model `"string"`
400	Invalid request payload **Example Value** Model `{`

```
"details": "string",
"error": "string",
"message": "string"
}
```

401 Unauthorized

Model
Example

Value {

```
"details": "string",
"error": "string",
"message": "string"
}
```

403 Forbidden - Admin privileges required

Model
Example

Value {

```
"details": "string",
"error": "string",
"message": "string"
}
```

500 Internal server error

Model
Example

Value {

```
"details": "string",
"error": "string",
"message": "string"
}
```

Annotation Threads

GET `/api/v1/cases/{case_id}/threads` Get threads by case ID
Retrieves all annotation threads associated with a specific case

Parameters **Try it out** **Name** **Description**

case_id *(path)* **required**

* `string`

case_id

Case ID

Responses **Response content type** **application/json**

Code **Description**

200 Threads retrieved successfully

Example Value Model

```
id : string ,
"is_active": true,
"participants": [
{
"joinedAt": "string",
"threadID": "string",
"userID": "string"
}
],
"priority": "high",
"resolved_at": "string",
"status": "open",
"tags": [
{
"id": "string",
"tag_name": "string",
"thread_id": "string"
}
],
"title": "string",
"updated_at": "string"
}
]
}
```

400 Invalid case ID

Example Model

Value {

```
  "details": "string",
```

Code **Description**

```
  "error": "string",
  "message": "string"
}
```

401 Unauthorized

Example Model

Value {

```
  "details": "string",
  "error": "string",
  "message": "string"
}
```

500 Internal server error

Example Model

Value {

```
  "details": "string",
  "error": "string",
  "message": "string"
}
```

POST `/api/v1/cases/{case_id}/threads` Create a new annotation thread

Creates a new annotation thread for a specific case and file

Parameters Try it out Name Description

case_id * *(path)*

`string`

request * case_id `object` *(body)*
required required

Example Value

Thread Creation {
Request

Case ID Model

```
 "file_id": "string",
"priority": "LOW",
"tags": [
 "string"
 ],
 "title": "string"
}
```

Parameter content type

Name Description

application/json

Responses **Response content type** **application/json**

Code **Description**

201 Thread created successfully

Example Value

"data": {
Model

```
{
"case_id": "string",
"created_at": "string",
"created_by": "string",
"file_id": "string",
"id": "string",
"is_active": true,
"participants": [
{
"joinedAt": "string",
"threadID": "string",
"userID": "string"
}
],
"priority": "high",
"resolved_at": "string",
```

```
"status": "open",
"tags": [
{
"id": "string",
"tag_name": "string",
"thread_id": "string"
```

400 Invalid request data

Value {

Example Model

```
"details": "string",
"error": "string",
"message": "string"
}
```

401 Unauthorized

Example Model

Value {

```
"details": "string",
"error": "string",
```

Code **Description**

```
"message": "string"
}
```

500 Internal server error

Example Model

Value {

```
"details": "string",
"error": "string",
"message": "string"
}
```

GET `/api/v1/cases/{case_id}/threads/{thread_id}` Get thread by ID

Retrieves a specific annotation thread by its ID

Parameters Try it out Name Description

case_id case_id *(path)* required Thread ID

* `string`

(path) required Case ID **thread_id** thread_id

* `string`

Responses Response content type **application/json**

Code **Description**

200 Thread retrieved successfully

Model
Example Value

```
{   "data": {
 "case_id": "string",
 "created_at": "string",
 "created_by": "string",
 "file_id": "string",
```

Code **Description**

```
"id": "string",

 "is_active": true,
 "participants": [
 {
 "joinedAt": "string",
 "threadID": "string",
 "userID": "string"
 }
 ],
 "priority": "high",
 "resolved_at": "string",
 "status": "open",
 "tags": [
 {
 "id": "string",
 "tag_name": "string",
```

400 Invalidthread ID

Model
Example

Value {

}

```
"details": "string",
"error": "string",
"message": "string"
```

401 Unauthorized

Value {

Model

Example

```
"details": "string",
"error": "string",
"message": "string"
}
```

404 Threadnotfound

Model

Example

Value {

```
"details": "string",
"error": "string",
"message": "string"
}
```

500 Internal servererror

Model

Example

Value {

```
"details": "string",
"error": "string",
"message": "string" }
```

GET `/api/v1/cases/{case_id}/threads/{thread_id}/participants`

Get thread participants

Retrieves all participants of a specific annotation thread

Parameters Try it out **Name Description**

case_id case_id **d** * string

* string *(path)*
required
(path) **thread_i** Thread ID
required
Case ID thread_id

Responses **Response content type** **application/json**

Code **Description**

200 Participants retrieved successfully

Example Value Model

```
{
  "data":
  [  {
    "joinedAt": "string",
    "threadID": "string",
    "userID": "string"
   }
  ]
 }
```

400 Invalid thread ID

Model
Example

Value

```
{
  "details": "string",
  "error": "string",
  "message": "string"
 }
```

401 Unauthorized

Example **Value** Model

Code **Description**

```
{
 "details": "string",
 "error": "string",
 "message": "string"
}
```

500 Internal server error

Model

Example

Value {

```
 "details": "string",
 "error": "string",
 "message": "string"
}
```

POST `/api/v1/cases/{case_id}/threads/{thread_id}/participants`
Add participant to thread

Adds a user as a participant to an annotation thread

Parameters Try it out Name Description

case_id * `string` *(path)* required Case ID case_id	**thread_id** * `string` *(path)* required Thread ID thread_id	
	request * required	Add Participant Request `object` *(body)* **Example** **Value** { Model

```
  "user_id": "string"
 }
```

Parameter content type

application/json

Responses **Response content type** **application/json**

Code **Description**

200 Participant added successfully

Example Value Model

```
"string"
```

400 Invalid request data

Example Value Model

```
{
  "details": "string",
  "error": "string",
  "message": "string"
}
```

401 Unauthorized

Example Value Model

```
{
  "details": "string",
  "error": "string",
  "message": "string"
}
```

500 Internal server error

Example Value Model

```
{
  "details": "string",
  "error": "string",
  "message": "string"
}
```

PUT `/api/v1/cases/{case_id}/threads/{thread_id}/priority` Update thread priority

Updates the priority of a specific annotation thread

Parameters Try it out

Name Description

case_id * Case ID

required

Name Description

(path)

`string`

(path)

Thread ID

thread_id

required

thread_id * `string`

case_id

Priority Request `string`

request *

required

Update Thread

any

(body)

Example Value

Model

Parameter content type

application/json

Responses **Response content type** **application/json**

Code	Description
200	Thread priority updated successfully

Example Value Model

```
"string"
```

400	Invalid request data

Example Value Model

```
{
  "details": "string",
  "error": "string",
  "message": "string"
}
```

401	Unauthorized

Example Value Model

```
{
  "details": "string",
  "error": "string",
  "message": "string" }
```

Code	Description
500	Internal server error

Example Value Model

```
{
  "details": "string",
  "error": "string",
  "message": "string"
}
```

PUT `/api/v1/cases/{case_id}/threads/{thread_id}/status` Update thread status

Updates the status of a specific annotation thread

Parameters Try it out Name Description

case_id case_id *(path)* required Thread ID

* `string`

(path) required Case ID **thread_id** thread_id

* `string`

Status Request `string`

request * any Model

required *(body)*

Update Thread **Example Value**

Parameter content type

application/json

Responses Response content type application/json

Code Description

200 Thread status updated successfully

Example Value `"string"`

Model

400 Invalid request data

Model

Example

Value
```
{
  "details": "string",
  "error": "string",
  "message": "string"
}
```

401 Unauthorized

Model

Example

Value
```
{
  "details": "string",
  "error": "string",
  "message": "string"
}
```

500 Internal server error

Model

Example

Value
```
{
  "details": "string",
  "error": "string",
  "message": "string"
}
```

GET `/api/v1/cases/{case_id}/threads/by-file/{file_id}` Get threads by file ID

Retrieves all annotation threads associated with a specific file

Parameters Try it out Name Description

case_id required Case ID case_id

* string

(path)
Name Description
string File ID

file_id * *(path)* required file_id

Responses Response content type **application/json**

Code **Description**

200 Threads retrieved successfully

Example Value "data": [{
Model

{

400 Invalid file ID

```
"case_id": "string",
"created_at": "string",
"created_by": "string",
"file_id": "string",
"id": "string",
"is_active": true,
"participants": [
{
"joinedAt": "string",
"threadID": "string",
"userID": "string"
}
],
"priority": "high",
"resolved_at": "string",
"status": "open",
"tags": [
{
"id": "string",
"tag_name": "string",
```

Value {

Model

Example

```
 "details": "string",
 "error": "string",
 "message": "string"
}
```

401 Unauthorized

Model

Example

Value {

```
 "details": "string",
```

Code **Description**

```
 "error": "string",
 "message": "string"
}
```

500 Internal server error

Model

Example

Value {

```
 "details": "string",
 "error": "string",
 "message": "string"
}
```

Thread Messages

GET `/api/v1/cases/{case_id}/threads/{thread_id}/messages`
Get messages by thread ID

Retrieves all messages in a specific annotation thread

Parameters **Try it out** **Name** **Description**

case_id * string *(path)* required Case ID	case_id	*(path)* required Thread ID
	thread_id * string	thread_id

Responses Response content type **application/json**

Code **Description**

200 Messages retrieved successfully

Example Value Model

```
{   "data": [
```

Code **Description**

```
{
"approvedAt": "string",
"approvedBy": "string",
"createdAt": "string",
"id": "string",
"isApproved": true,
"mentions": [
{
"createdAt": "string",
"mentionedUserID": "string",
"messageID": "string"
}
],
"message": "string",
"parentMessageID": "string",
"reactions": [
{
"createdAt": "string",
"id": "string",
"messageID": "string",
"reaction": "string",
```

400 Invalid thread ID

Example Model

Value {

```
"details": "string",
"error": "string",
"message": "string"
}
```

401 Unauthorized

Example Model

Value {

```
"details": "string",
"error": "string",
"message": "string"
}
```

500 Internal server error

Example Model

Value {

```
"details": "string",
"error": "string",
"message": "string"
}
```

POST `/api/v1/cases/{case_id}/threads/{thread_id}/messages`
Send a message in a thread

Creates a new message in a specific annotation thread
Parameters Try it out Name Description

case_id * `string` *(path)* **required** Case ID

case_id

thread_id * `string` *(path)* **required** Thread ID

thread_id

request * **required** Send Message Request

any *(body)* **Example**

Value

`string`

Model

Parameter content type

application/json

Responses **Response content type** **application/json**

Code **Description**

201 Message sent successfully

Example Value Model

```
{   "data": {
 "approvedAt": "string",
 "approvedBy": "string",
 "createdAt": "string",
 "id": "string",
 "isApproved": true,
 "mentions": [
 {
 "createdAt": "string",
 "mentionedUserID": "string",
 "messageID": "string"
 }
 ],
 "message": "string",
 "parentMessageID": "string",
 "reactions": [
```

Code **Description**

```
" " " i "
```

400 Invalid request data

```
{
 "createdAt": "string",
 "id": "string",
 "messageID": "string",
 "reaction": "string",
```

Value {

Model

Example

```
  "details": "string",
  "error": "string",
  "message": "string"
}
```

401 Unauthorized

Model

Example

Value {

500 Internal server error

```
  "details": "string",
  "error": "string",
  "message": "string"
}
```

Value {

Model

Example

```
  "details": "string",
  "error": "string",
  "message": "string"
}
```

GET `/api/v1/cases/{case_id}/threads/{thread_id}/messages/{message_id}` Get message by ID

Retrieves a specific message by its ID

Parameters Try it out Name Description

case_id * `string` *(path)*

required
Case ID

case_id

thread_id Thread ID

* `string` thread_id

required

Name Description *(path)*

message_ **id** * `string` Message ID

(path) message_id

required

Responses **Response content type** **application/json**

Code **Description**

200 Message retrieved successfully

Example Value Model

```
"data": {
{
  "approvedAt": "string",
  "approvedBy": "string",
  "createdAt": "string",
  "id": "string",
  "isApproved": true,
  "mentions": [
  {
  "createdAt": "string",
  "mentionedUserID": "string",
  "messageID": "string"
  }
  ],
  "message": "string",
  "parentMessageID": "string",
  "reactions": [
  {
  "createdAt": "string",
  "id": "string",
  "messageID": "string",
  "reaction": "string",

  "userID": "string"
```

400 Invalid message ID

Example **Value** {

Model

```
 "details": "string",
 "error": "string",
 "message": "string"
}
```

401 Unauthorized

Model

Example Value

Code **Description**

```
{
 "details": "string",
 "error": "string",
 "message": "string"
}
```

404 Message not found

Model

Example

Value {

```
 "details": "string",
 "error": "string",
 "message": "string"
}
```

500 Internal server error

Model

Example

Value {

```
 "details": "string",
 "error": "string",
 "message": "string"
}
```

PUT`/api/v1/cases/{case_id}/threads/{thread_id}/messages/{message_id}/approve`
Approve a message

Approves a specific message in a thread

Parameters Try it out Name Description

case_id *

`string`

(path)
required
Case ID

case_id

thread_id * `string`

(path)
required
Thread ID

thread_id

message_id *

`string`

(path)
required
Message ID

message_id

Responses **Response content type** **application/json**

Code **Description**

200 Message approved successfully

Example Value Model

```
"string"
```

400 Invalid message ID

Example Value Model

```
{
  "details": "string",
  "error": "string",
  "message": "string"
}
```

401 Unauthorized

Example Value Model

```
{
```

```
"details": "string",
"error": "string",
"message": "string"
}
```

500 Internal server error

Model

Example

Value {

```
"details": "string",
"error": "string",
"message": "string"
}
```

POST `/api/v1/cases/{case_id}/threads/{thread_id}/messages/{message_id}/mentions` Add mentions to message

Adds user mentions to a specific message

Parameters Try it out

Name Description

required

Thread ID

case_id *

thread_id

request *

`string`

required

Add Mentions Request

`(path)`

required

Case ID

message `object`

case_id

_id *

`(body)`

`string`

Example

`(path)`

Value {

thread_id

required

Message ID

* `string`

Model

message_id

`(path)`

```
"mentions": [
"string"
]
}
```

Parameter content type

application/json

Responses Response content type application/json

Code Description

200 Mentions added successfully

Example Value Model

```
"string"
```

400 Invalid request data

Example Value Model

```
{
  "details": "string",
  "error": "string",
  "message": "string" }
```

Code Description

401 Unauthorized

Example Value Model

```
{
  "details": "string",
  "error": "string",
  "message": "string"
}
```

500 Internal server error

Example Value Model

```
{
  "details": "string",
  "error": "string",
  "message": "string"
}
```

DELETE `/api/v1/cases/{case_id}/threads/{thread_id}/messages/{message_id}/reactions`
Remove reaction from message

Removes a user's reaction from a specific message

Parameters Try it out Name Description

Name	Description
case_id * string *(path)* required	Case ID case_id
thread_id * string *(path)* required	Thread ID thread_id
message_id * string *(path)* required	Message ID message_id

Responses **Response content type** **application/json**

Code	Description
200	Reaction removed successfully **Example Value** Model `"string"`
400	Invalid message ID

Model
Example

Value {

```
  "details": "string",
  "error": "string",
  "message": "string"
}
```

401 Unauthorized

Model
Example

Value {

```
  "details": "string",
  "error": "string",
  "message": "string"
}
```

500 Internal server error

Model
Example

Value {

```
  "details": "string",
  "error": "string",
  "message": "string"
}
```

POST `/api/v1/cases/{case_id}/threads/{thread_id}/messages/{message_id}/reactions` Add reaction to message

Adds a reaction to a specific message

Parameters Try it out

Name Description

case_id * `string` *(path)*	required Case ID case_id
thread_id * `string`	

(path) **required** Thread ID thread_id	`string` ***(path)*** **required** Message ID message_id	Add Reaction Request `object` ***(body)*** **Example Value** { Model
message _id *	**request** * **required**	

```
 "reaction": "string"
}
```

Parameter content type

application/json

Responses **Response content type** **application/json**

Code **Description**

200 Reaction added successfully

Example Value

```
"string"
```

Model

400 Invalid request data

Model

Example Value {

```
 "details": "string",
"error": "string",
"message": "string" }
```

Code **Description** {

401 Unauthorized

Example Value Model

```
"details": "string",
"error": "string",
"message": "string"
}
```

500 Internal server error

Example Model

Value {

```
"details": "string",
"error": "string",
"message": "string"
}
```

GET `/api/v1/cases/{case_id}/threads/{thread_id}/messages/{message_id}/replies` Get message replies

Retrieves all replies to a specific message

Parameters **Try it out** **Name** **Description**

case_id *
`string`
(path)
required
Case ID

case_id

thread_id *
`string`
(path)
required
Thread ID

thread_id

message_id *
`string`
(path)
required
Message ID

message_id

Responses **Response content type** **application/json**

Code **Description**

200 Repliesretrievedsuccessfully

Model

Example Value

```
{
 "data": [  {

  "approvedAt": "string",
  "approvedBy": "string",
  "createdAt": "string",
  "id": "string",
  "isApproved": true,
  "mentions": [
  {
  "createdAt": "string",
  "mentionedUserID": "string",
  "messageID": "string"
  }
  ],
  "message": "string",
  "parentMessageID": "string",
  "reactions": [
  {
  "createdAt": "string",
  "id": "string",
  "messageID": "string",

  "reaction": "string",
```

400 Invalidmessage ID

Model

Example

Value

```
{

  "details": "string",
  "error": "string",
  "message": "string"
 }
```

401 Unauthorized

Model

Example

Value {

```
  "details": "string",
  "error": "string",
  "message": "string"
 }
```

500 Internal servererror

Model

Example

Value {

```
  "details": "string",
 "error": "string",
 "message": "string" }
```

DIAGRAMS

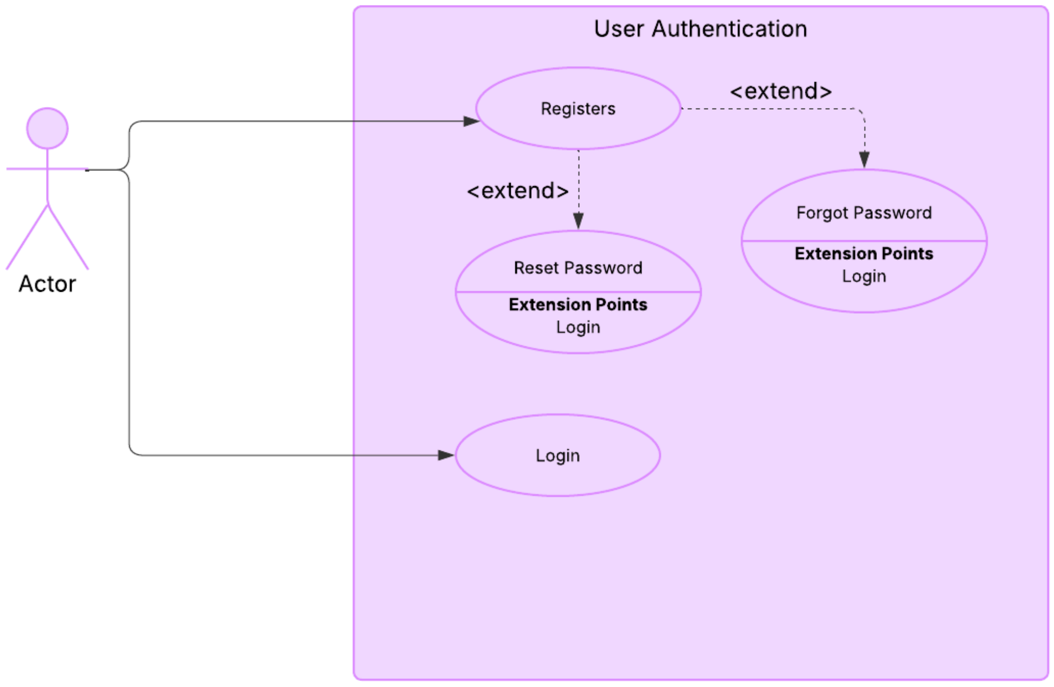
Domain Model

This section presents the domain model for AEGIS, which identifies and defines the key entities, relationships, and constraints within the system's problem space. It serves as a conceptual blueprint that informs system structure, data persistence, and service interactions. The model captures core components such as users, cases, evidence, annotation threads, and access roles.

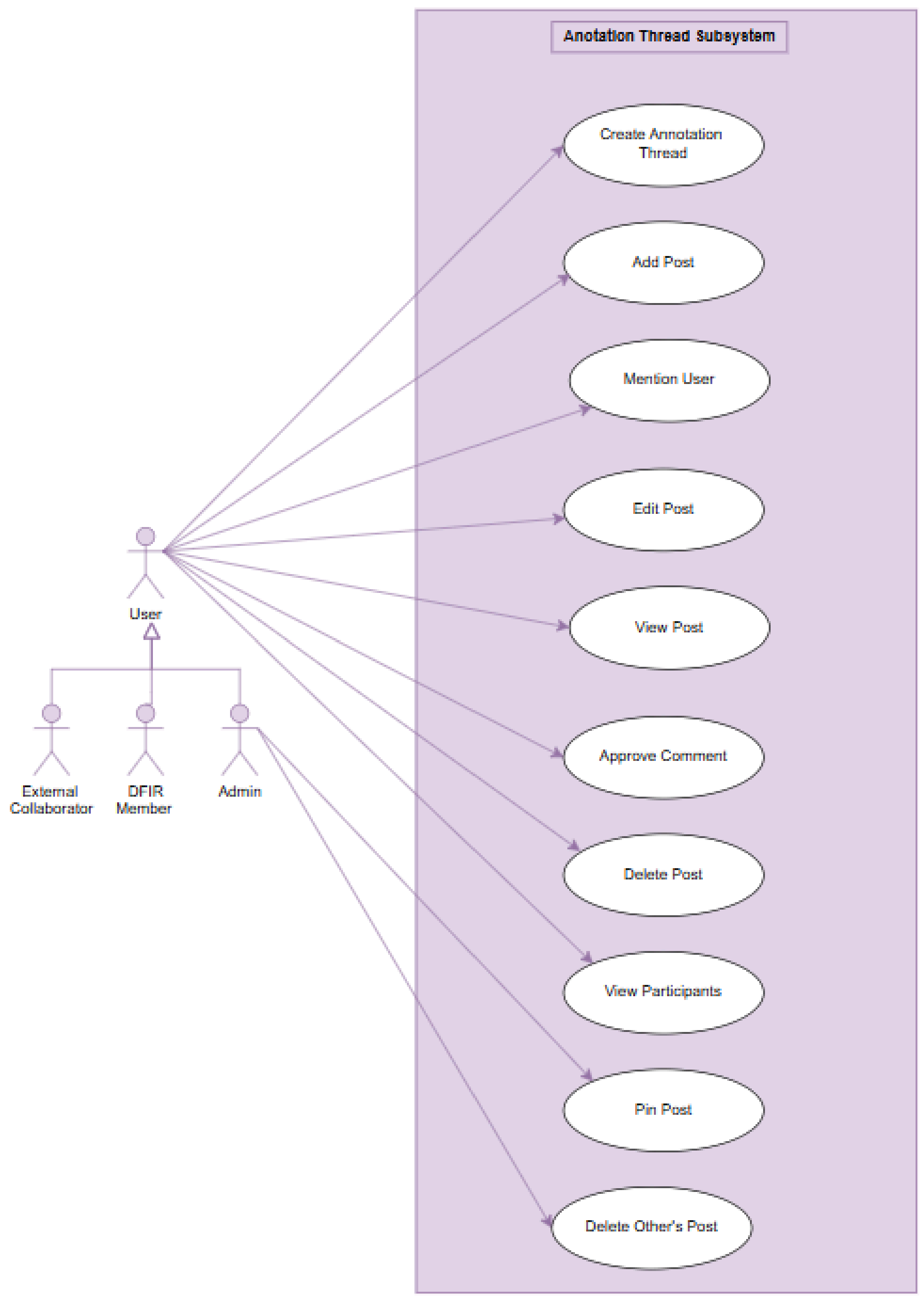
USE CASE DIAGRAMS

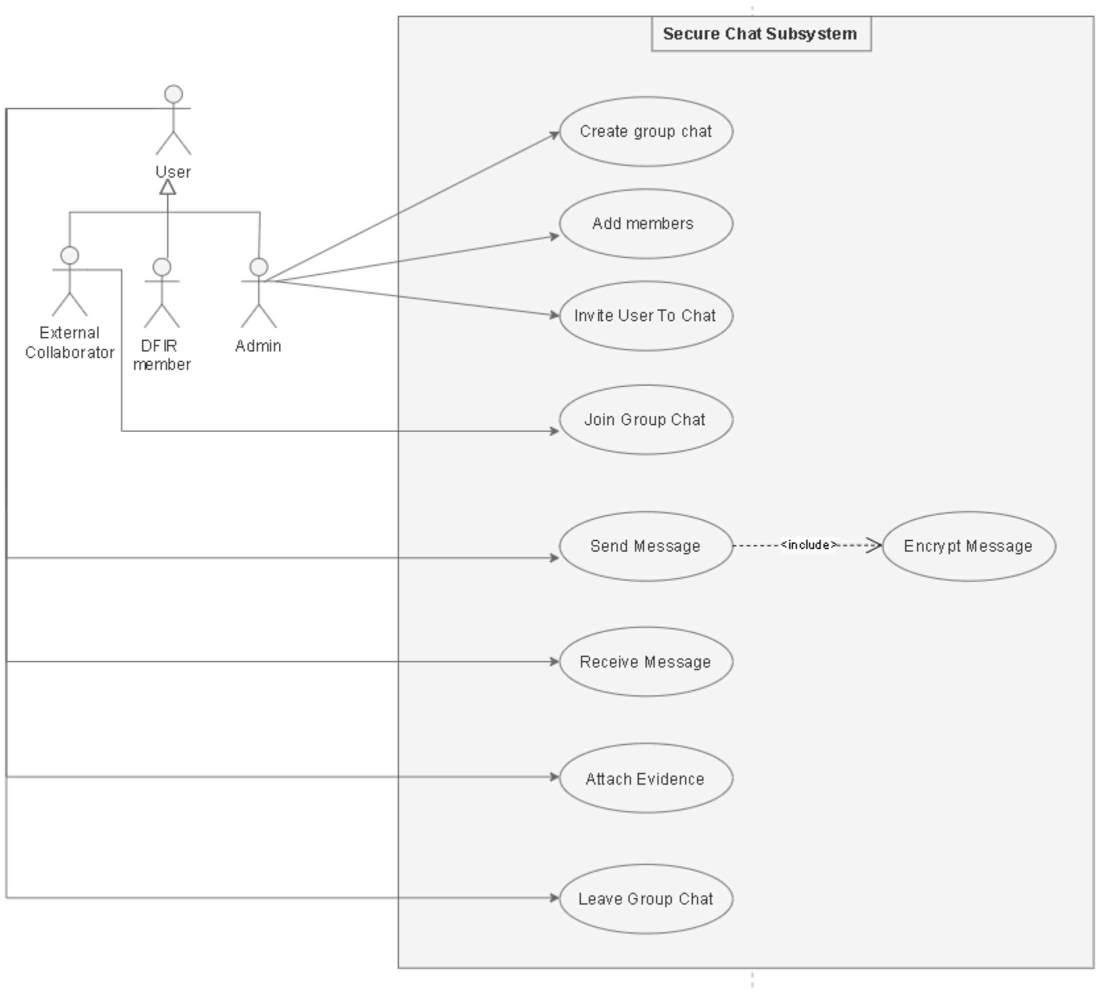
The use case diagrams provide a high-level visual representation of the system's primary functionalities and the interactions between users and system components. Organized by system modules (e.g., evidence management, collaboration, access control), these diagrams help clarify user expectations and guide functional decomposition.

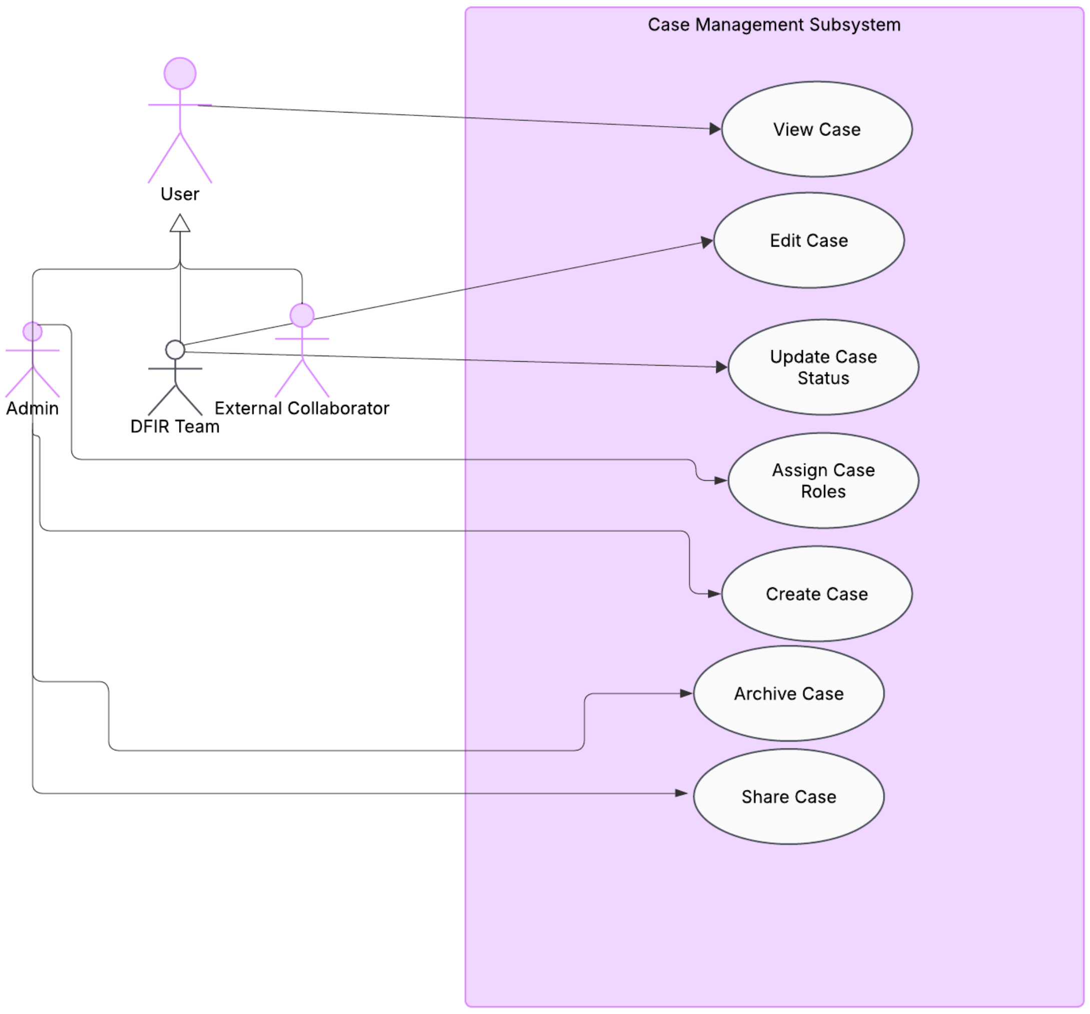
User Authentication

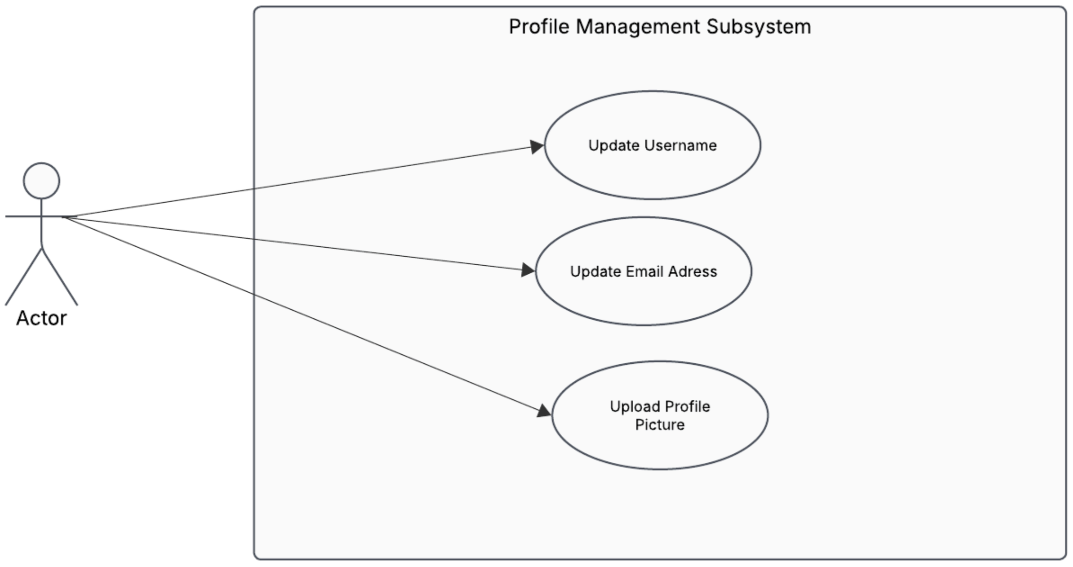


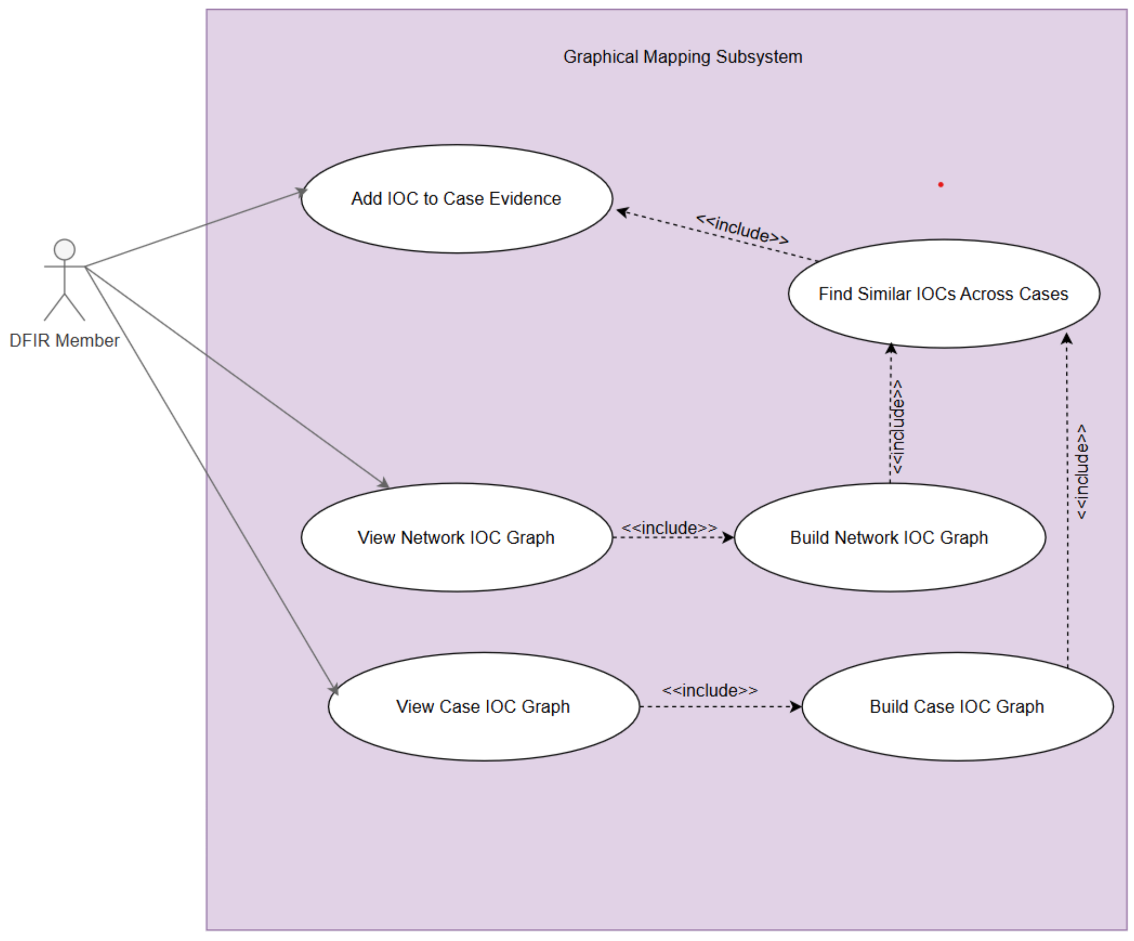
Annotation Thread Subsystem

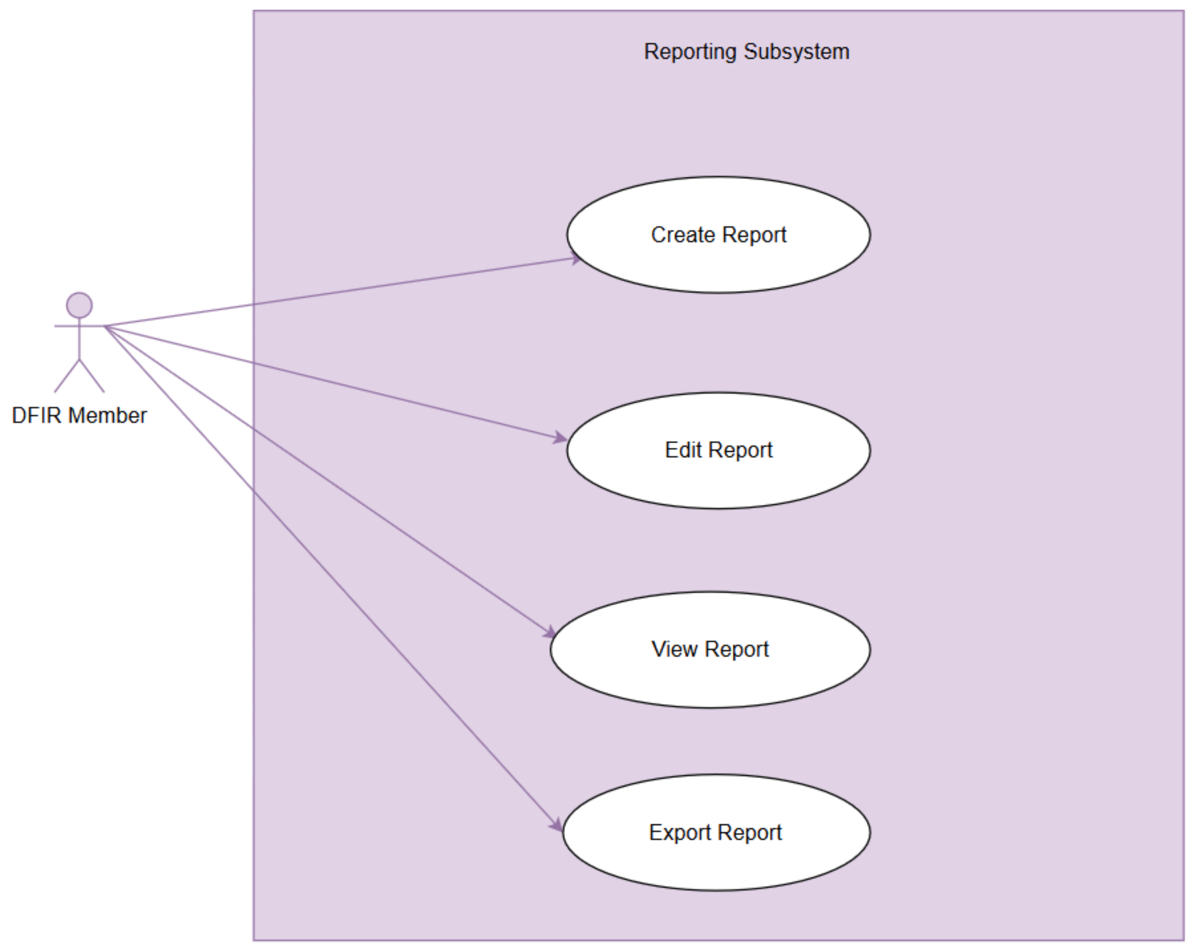

Anotation Thread Subsystem
Create Annotation Thread
Add Post
Mention User
Edit Post
View Post
Approve Comment
Delete Post
View Participants
Pin Post
Delete Other's Post
User
External Collaborator
DFIR Member
Admin

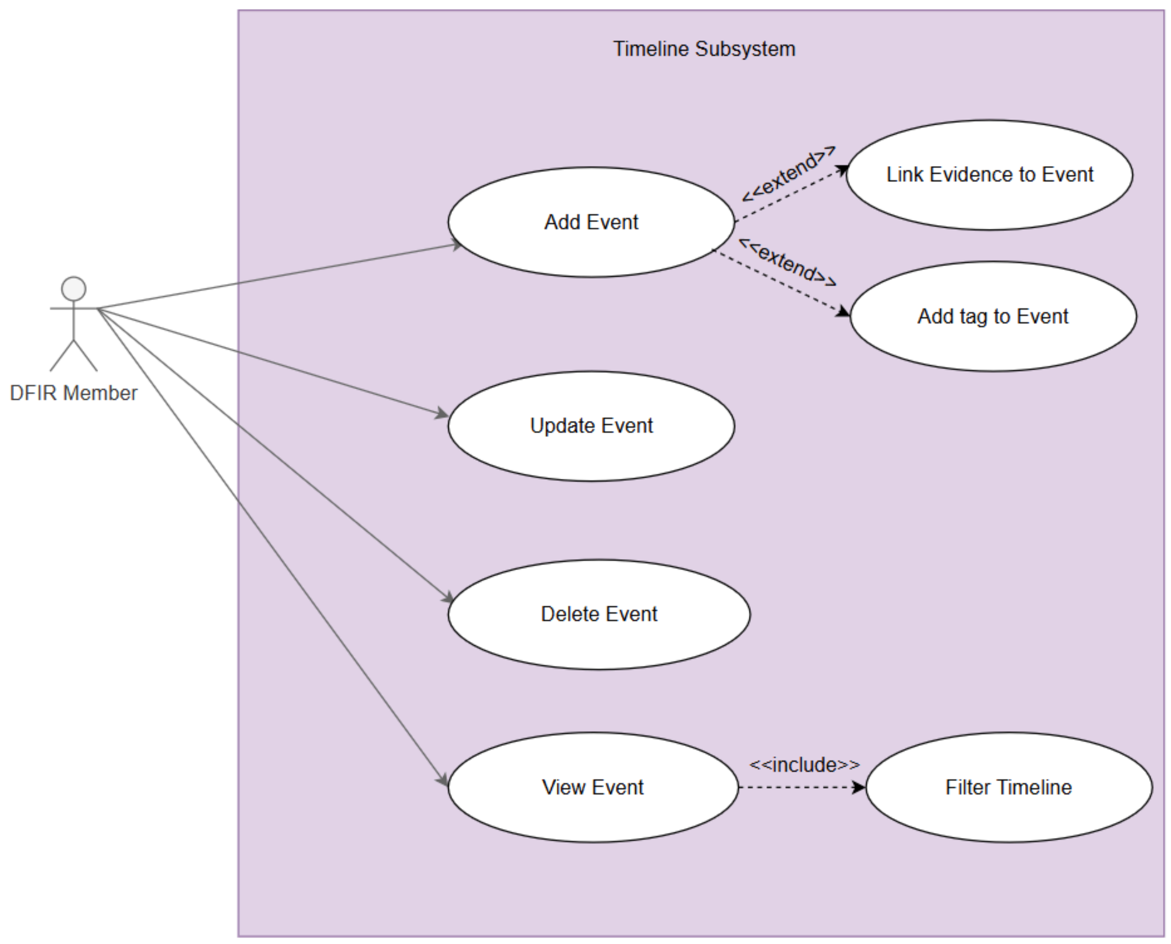

Secure Chat Subsystem
User
External Collaborator
DFIR member
Admin
Create group chat
Add members
Invite User To Chat
Join Group Chat
Send Message
<include>
Encrypt Message
Receive Message
Attach Evidence
Leave Group Chat


Case Management Subsystem
User
Admin
DFIR Team
External Collaborator
View Case
Edit Case
Update Case Status
Assign Case Roles
Create Case
Archive Case
Share Case


Profile Management Subsystem
Actor
Update Username
Update Email Adress
Upload Profile Picture

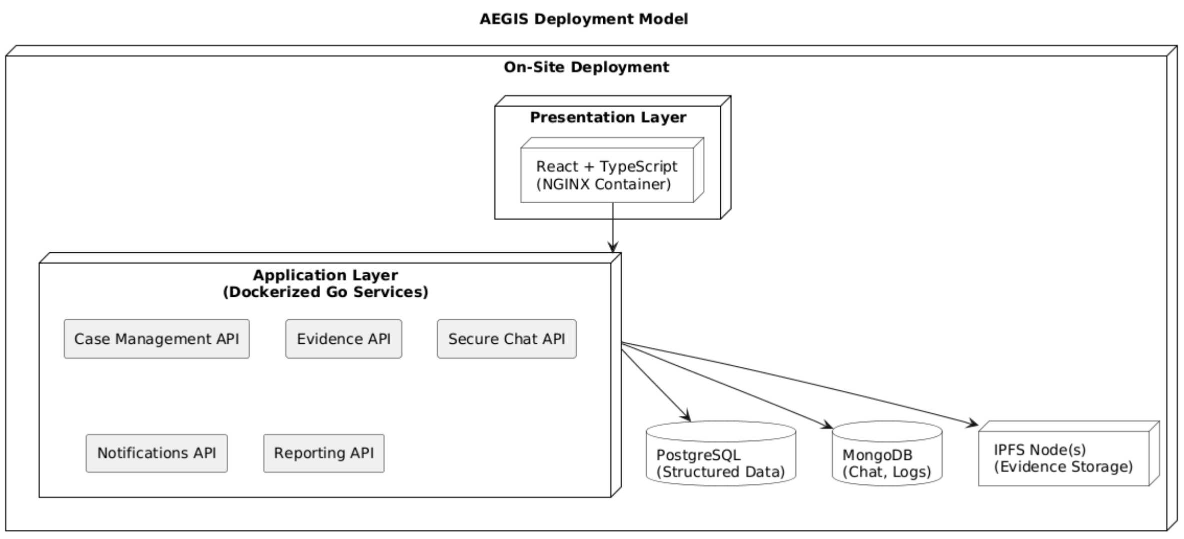

Graphical Mapping Subsystem
DFIR Member
Add IOC to Case Evidence
<<include>>
Find Similar IOCs Across Cases
<<include>>
<<include>>
View Network IOC Graph
<<include>>
Build Network IOC Graph
View Case IOC Graph
<<include>>
Build Case IOC Graph


Reporting Subsystem
DFIR Member
Create Report
Edit Report
View Report
Export Report

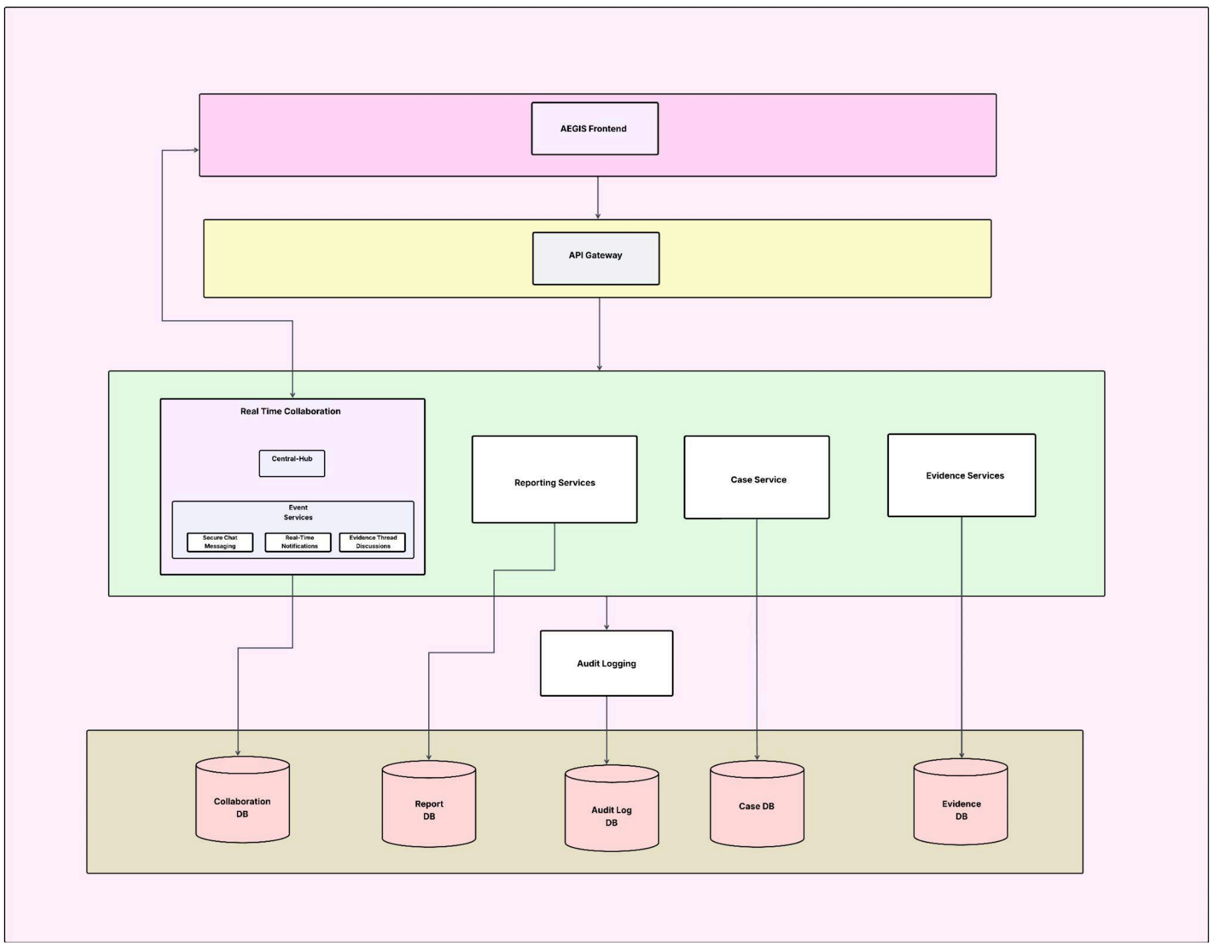


DEPLOYMENT MODEL

The deployment model describes how the AEGIS platform is packaged, deployed, and maintained in production. It covers container-based deployment, environment configurations, CI/CD pipelines, scaling strategies, and deployment to a private or cloud-based infrastructure for secure, controlled access.



ARCHITECTURAL DIAGRAM



LIVE DEPLOYED SYSTEM

This section documents the publicly or privately accessible live deployment of the AEGIS platform. It includes the system URL, access credentials for demo purposes, deployment status, and monitoring tools. This live instance demonstrates the system’s readiness for real-world forensic investigation workflows.

Will be updated towards DEMO 4