Project

Phase 1

EventFlow

Desenvolvimento de Software Seguro

1230185 – Francisco Pereira

1240598 – Rafael Ferreira

1240600 – Ricardo Cruz

1240601 – Xavier Ricarte

Table of Contents

Table of Figures

Indice of Table

1 Project Description

This report provides an in-depth explanation of the process involved in the Event Management Platform's development and security framework. The platform will provide end-to-end event management to allow different user roles such as Administrators, Event Managers, Clients, Moderators, and Suppliers to perform critical actions such as creating, managing, attending, and moderating events in a secure, permissioned setting.

Throughout this report, various elements of the system's design and deployment are examined with a focus on threat detection, risk assessment, and mitigation of security vulnerabilities. These are aligned with Secure Software Development Lifecycle (SSDLC) procedures, which render the platform extremely confidential, integral, and available.

During this initial phase, we defined both non-functional and functional requirements, along with the necessary use cases determining the roles and interactions between users and the system. To provide a clearer view of the system structure and behavior, a number of Data Flow Diagrams (DFDs) were created at Level 2, showing significant processes, trust boundaries, authentication procedures, and database interactions.

This paper is the foundation for secure and scalable construction of the platform and provides an important glimpse into its design, security position, and operational pipeline.

2 Domain Model

This UML class diagram represents the architecture of an **Event Management Platform**, designed using **Domain-Driven Design (DDD)** principles. The system is modularized into four main bounded contexts (packages), each responsible for a distinct part of the application's domain logic.

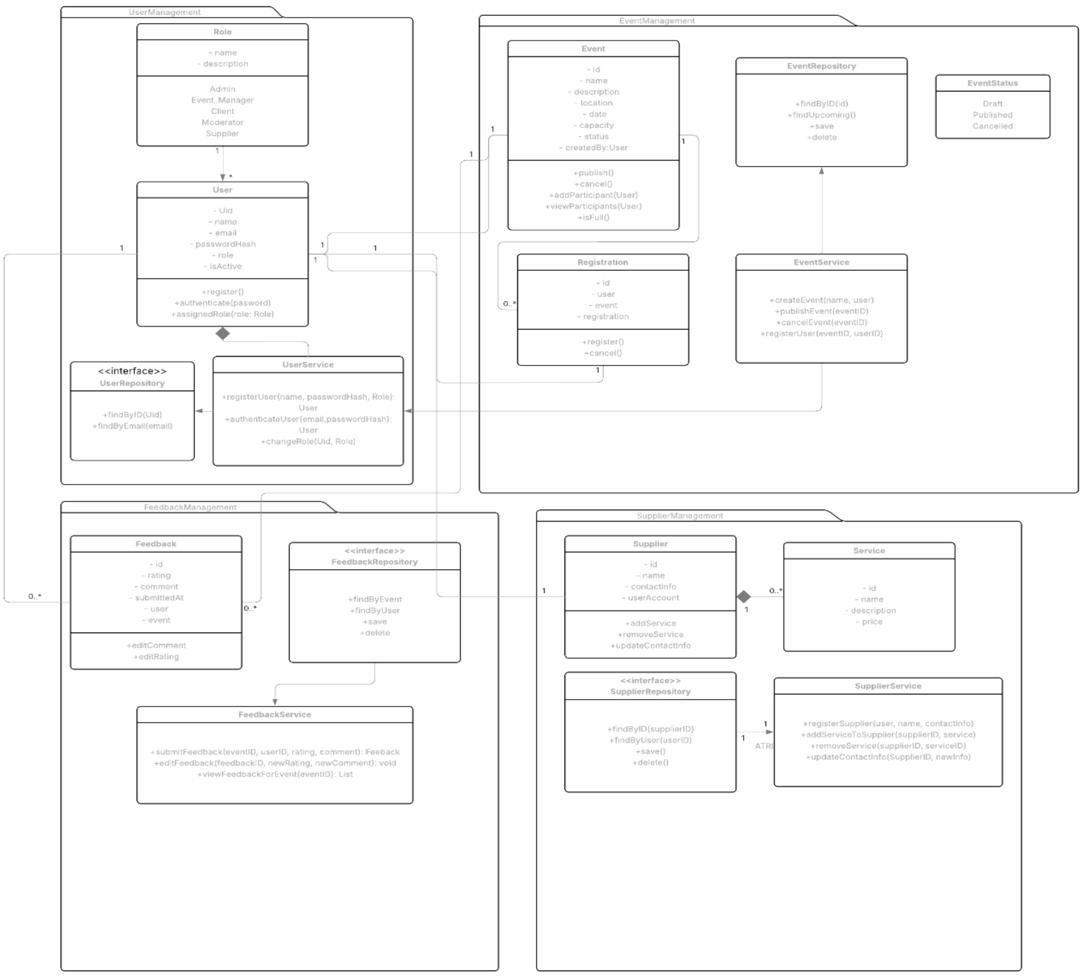


Figure 1 - Domain Model

3 Requirements and Use Cases

This section presents the fundamental requirements and use cases that support the development of the event management platform.

The requirements define the essential functional and non-functional characteristics to ensure the correct operation, security, and usability of the system.

The use cases describe, in a practical and structured way, the interactions between the different types of users and the platform, illustrating the main operational flows and the expected behavior.

3.1 Functional Requirements

Functional requirements describe the functions and services that the system has.

Admin
Total management of the application;
Management of the remaining users including their role and permissions;
Monitorization of events and statistics related to it;
Manage event categories

Table 1 - Admin Functional Requirements

Event Manager
Create new events, providing necessary details
Edit and deletion/delaying a event
View feedback related to their managed events for quality improvement
Management of the event inscriptions and attendees;

Table 2 - Event Manager Functional Requirements

Client
Browse and search available events by categories, dates, or keywords.
Register (inscribe) for events and manage their inscriptions
View their registered events and participation history
Evaluation and commentaries on the events

Table 3 - Client Functional Requirements

Moderator
Approval or rejection of events submitted by event managers
Moderation of the commentaries and evaluations
Report abusive content or inappropriate behavior detected in events or feedback

Table 4 - Moderator Functional Requirements

Supplier/Partner
Creates a profile approved by the administrator in order to show the products/services that will be selling
Can only communicate directly with the event managers
Management of their own product inside the application

Table 5 - Supplier/Partner Functional Requirements

3.2 Non-Functional Requirements

This section details the non-functional requirements that the event management platform must fulfill to ensure its reliability, security, performance, and overall user experience.

Non-functional requirements define the quality attributes of the system, establishing essential criteria such as availability, scalability, maintainability, usability, and supportability, which are critical to guarantee the platform’s success and sustainability.

3.2.1 Security Requirements

Role-Based Access Control (RBAC) – This type of access control ensures that different role types have specific permissions, this way we can ensure that we apply the principle of **least privileged access:**

- **Admin** – Full access to all system features, including user and role management, event category creation, monitoring of system statistics, and approval of supplier services.
- **Event Manager** – Ability to create, edit, reschedule, and cancel events; manage event registrations; and view feedback related to their events.
- **Moderator** – Ability to approve or reject events created by Event Managers, moderate client feedback, and manage reports of inappropriate content.
- **Client** – Ability to search for events, register for events, submit feedback and ratings, and manage their event participation history.
- **Supplier/Partner** – Ability to create and manage their service profiles, submit services for approval, and interact directly with Event Managers regarding service provision.

Authentication - strong authentication mechanisms (e.g., multi-factor authentication, OAuth 2.0) to ensure only authorized users can access the platform.

Data Encryption - All sensitive user data (e.g., passwords, payment information) should be encrypted both at rest and in transit (e.g., using HTTPS, AES encryption).

Session Management - Implement secure session handling, with automatic session expiration after a period of inactivity and support for session revocation.

Data Integrity - Ensure data integrity by implementing measures like input validation, error checking, and logging to prevent data corruption or unauthorized modification.

3.2.2 Usability Requirements

User-friendly interface - The platform should have an intuitive, easy-to-navigate interface, with clear labels, buttons, and tooltips to guide users.

Responsive interface - The frontend should be fully responsive and work well on all devices (desktop, tablet, mobile).

Minimal learning curve - New users should be able to understand how to use the platform with minimal training. Features should be intuitive, with in-app tutorials or tooltips when needed.

Error handling - Clear error messages should be displayed if the user attempts an invalid action (e.g., registration failure, incorrect login credentials), providing helpful hints for resolution.

3.2.3 Functionality Requirements

Authentication Services: Integration with external authentication providers (e.g., OAuth 2.0, Google, Microsoft) to allow users to authenticate securely using their existing accounts.

Notification Services: Integration with email and SMS providers (e.g., SendGrid, Twilio) to send notifications to users regarding event updates, registration confirmations, and feedback submissions.

Calendar Services: Allow users to export event details directly to their personal calendars (e.g., Google Calendar, Outlook), improving participation rates and user satisfaction.

Location Services: Integration with map services (e.g., Google Maps) to provide users with accurate event location details, directions, and transportation options.

Backup and Recovery Services: Integration with cloud backup services to ensure secure and automatic data backups, enabling fast recovery in case of system failure or data corruption.

3.2.4 Performance Requirements

Response time - The platform should provide quick responses to user requests. For example, page loading times should be under 3 seconds, and API requests should respond within 1 second for most operations.

Scalability - The system should be able to scale horizontally to handle increasing numbers of users and events. This includes support for cloud deployment and load balancing.

Database efficiency - The platform should ensure efficient database queries, using indexing and optimized queries to support fast data retrieval.

Concurrency – The system should be able to handle multiple users interacting with the plataform simultaneously without performance degradation

Fault Tolerance - The system should gracefully handle failures, including database crashes or API timeouts, and ensure that users experience minimal disruption.

Availability - The platform should be highly available, with a target uptime of 99.9% or higher, ensuring that users can access the platform whenever needed.

3.2.5 Supportability Requirements

Maintainability - The codebase should be modular and well-documented to allow easy updates, bug fixes, and feature additions. Use of consistent coding practices and comments is essential.

Monitoring and Logging - Implement monitoring tools to track the health of the system (e.g., server uptime, database performance) and log critical events (e.g., errors, failed login attempts, system alerts) for troubleshooting.

Backup and Restore - Regular backups of critical data (e.g., user information, event details) should be taken, with a documented process for restoring the platform in case of data loss.

Automated Testing and Deployment - Implement automated testing (unit tests, integration tests) and continuous deployment pipelines to ensure that new updates do not introduce bugs or vulnerabilities.

Documentation - Maintain detailed documentation for developers, system administrators, and end-users. This should include setup instructions, API documentation, and user manuals.

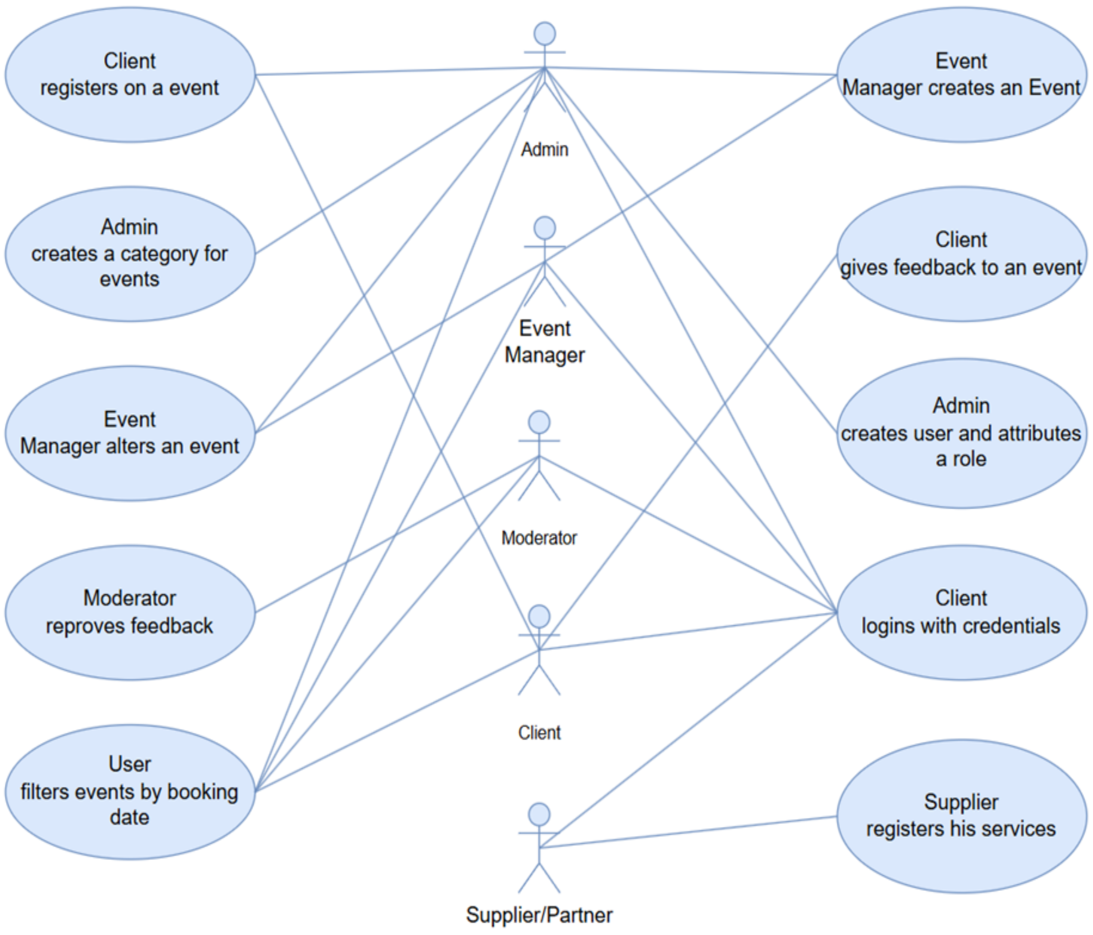
Support for new features - The system should be flexible enough to add new features (e.g., integrating additional payment providers or adding new user roles) without significant changes to the core architecture.

3.3 Use Cases

The actors for this system are:

- Admin
- Event Manager
- Moderator
- Client
- Supplier/Partner

Diagram


Client registers on a event
Admin creates a category for events
Event Manager alters an event
Moderator reproves feedback
User filters events by booking date
Admin
Event Manager
Moderator
Client
Supplier/Partner
Event Manager creates an Event
Client gives feedback to an event
Admin creates user and attributes a role
Client logins with credentials
Supplier registers his services

4 Threat Modelling Process

The Threat Modelling Process is an important secure software development practice. Through this process, the identification, classification, and countermeasures of threats that may undermine the security of an application can be done. In the case of the event management platform, we use the STRIDE model to determine possible threats, and the DREAD model to evaluate and prioritize the risks of these threats. In addition, according to the threats found, we suggest mitigations so that the platform is robust and safe from attacks.

4.1 System Information

- **Application Name:** EventFlow

- **Application Version:** 1.0

- **Description:** The event management platform allows you to create, moderate and participate in events. There are five types of users:
    - Admin
    - Event Manager
    - Moderator
    - Client
    - Supplier/Partner

- **Participants:** Francisco Pereira, Rafael Ferreira, Ricardo Cruz, Xavier Ricarte

- **Reviewer:** Nuno Pereira – NAP

4.2 External Dependencies

External dependencies are crucial for the secure and efficient operation of the web application. These include:

ID	Description
1	Cloud infrastructure supports both the front-end and back-end, ensuring scalability and availability.
2	MySQL Database located in the cloud, utilized for storing user information, event specifics, and feedback.
3	External APIs for event information synchronization and payment processing.
4	Authentication and authorization services (OAuth2 or JWT tokens).

Table 6 - External Dependencies

4.3 Entry Points

Entry points refer to the interfaces that allow external users or attackers to engage with the system. Locating and safeguarding these points aids in shielding the system from unapproved access.

ID	Name	Description	Trust Levels
1	Homepage	The landing page where users can view events and general platform information.	Admin, Event Manager, Client, Supplier
2	Login Form	Form used by users to authenticate themselves before accessing restricted resources.	Client, Supplier, Event Manager
3	Event Creation	Allows Event Managers to create new events and manage event details.	Event Manager
4	Event Registration	Allows Clients to register for events.	Client
5	Feedback	Allows users to submit feedback for events.	Client, Event Manager

Table 7 - Entry Points

4.4 Exit Points

Exit points are locations where data exits the system, whether in response to user actions or as exports to different systems.

ID	Name	Description	Potential Vulnerabilities
1	Event Data	Data provided to users, including event details, dates, and registration info.	Information Disclosure, SQL Injection
2	Feedback	Feedback submitted by clients on events.	XSS, SQL Injection
3	Event Statistics	Data sent to event organizers for tracking performance and registration.	Data Tampering, Insecure Data Storage

Table 8 - Exit Points

4.5 Assets

Assets are valuable items within the system that require safeguarding against threats. These may be physical (tangible) or non-physical (data, user credentials).

ID	Name	Description	Trust Levels
1	User Credentials	Includes usernames and passwords used for authentication.	Admin, Event Manager, Client, Supplier
2	Event Data	Information regarding events, including descriptions, schedules, and locations.	Admin, Event Manager, Client
3	Feedback	Reviews and comments left by clients on events.	Client, Event Manager
4	Payment Data	Sensitive payment information submitted during event registration.	Client, Admin

Table 9 – Assets

4.6 Trust Levels

Trust levels determine the access rights granted to various users or external parties engaged with the system.

ID	**Name**	**Description**
1	Admin	Full access to all system functionalities, including user management and event control
2	Event Manager	Can create and manage events, track registrations, and view feedback
3	Client	Can register for events, submit feedback, and view event details
4	Supplier/Partner	Can submit services for events but has limited interaction with the platform

Table 10 - Trust Levels

5 Data Flow Diagram

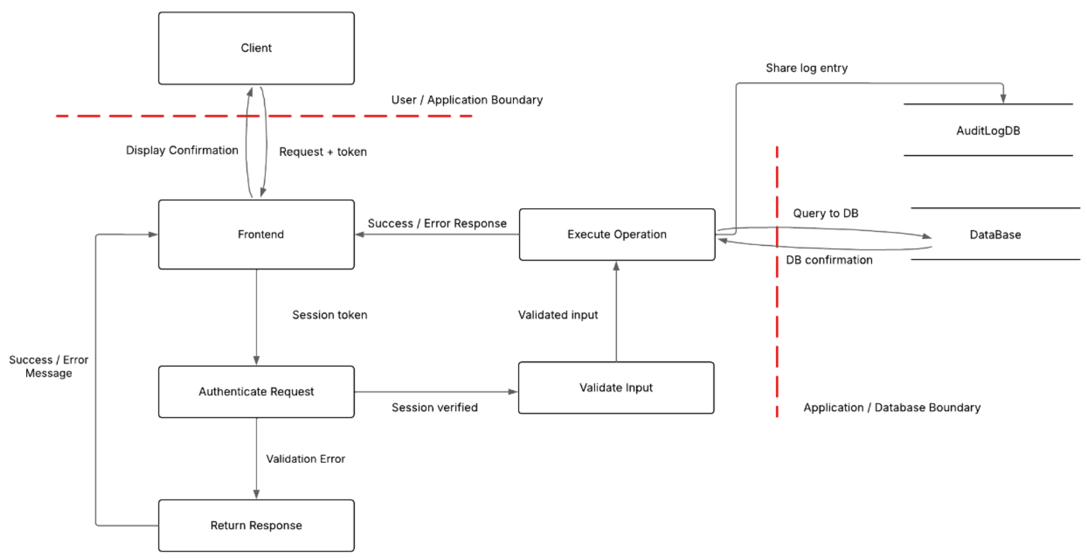


Figure 2 - Data Flow Diagram

5.1 Event Manager creates an Event

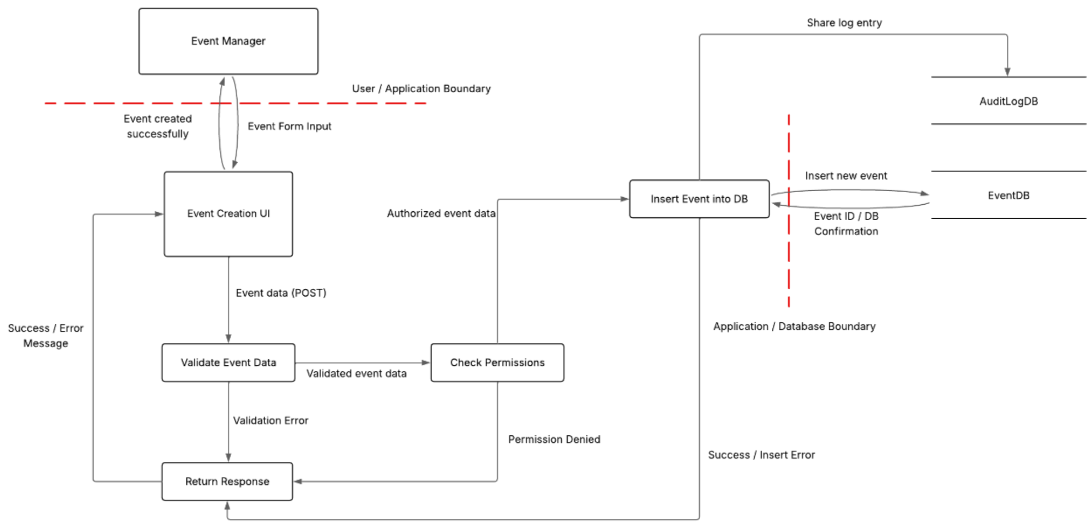


Figure 3 - DFD "Event Manager creates an Event"

5.2 Client gives feedback to an event

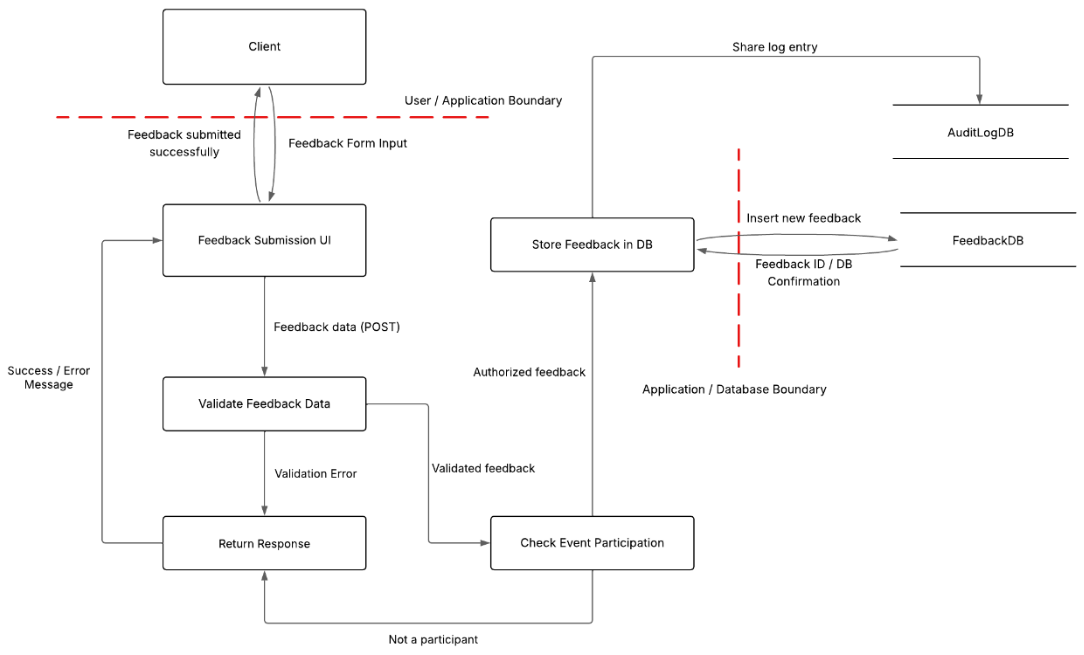


Figure 4 - DFD "Client gives feedback to an event"

5.3 Admin creates user and attributes a role

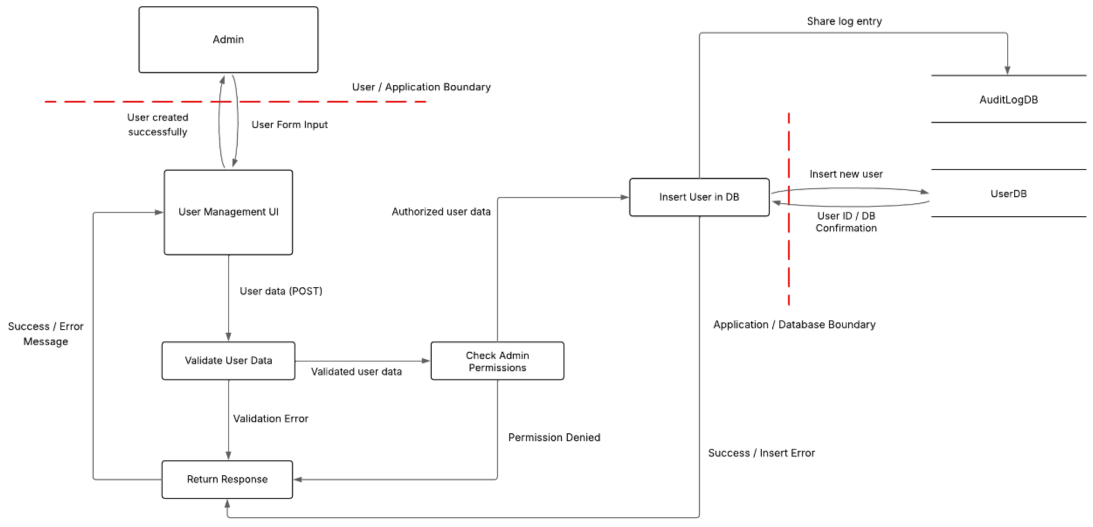


Figure 5 - DFD "Admin creates user and attributes a role"

5.4 Client logins with credentials

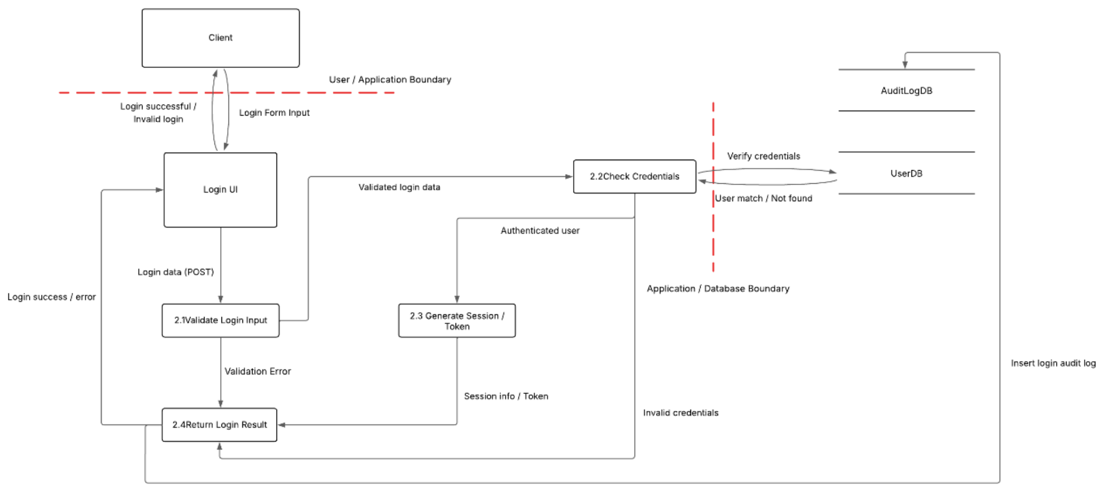


Figure 6 - DFD "Client logins with credentials"

5.5 Supplier registers his services

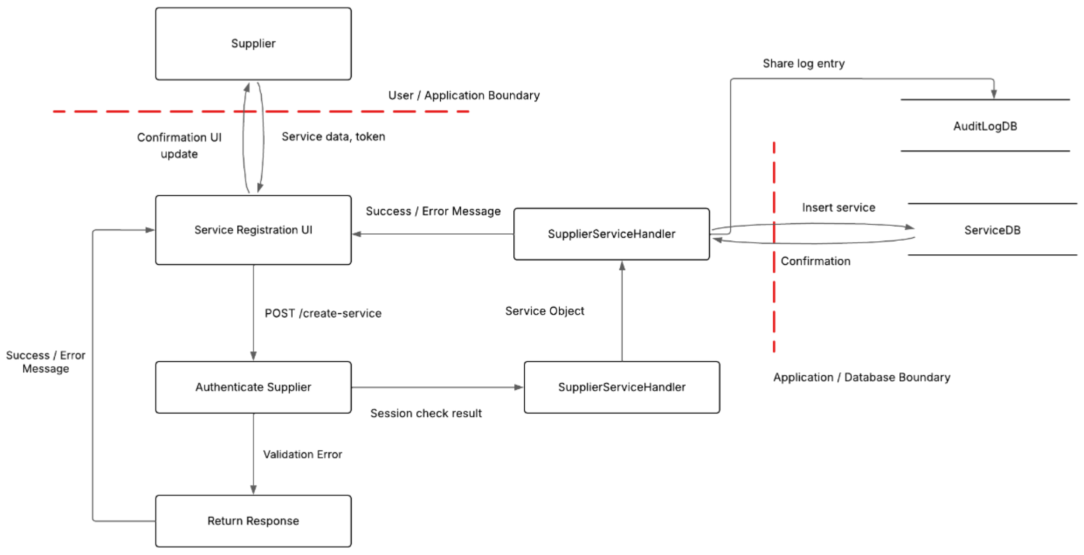


Figure 7 - DFD "Supplier registers his services"

5.6 Moderator reproves feedback

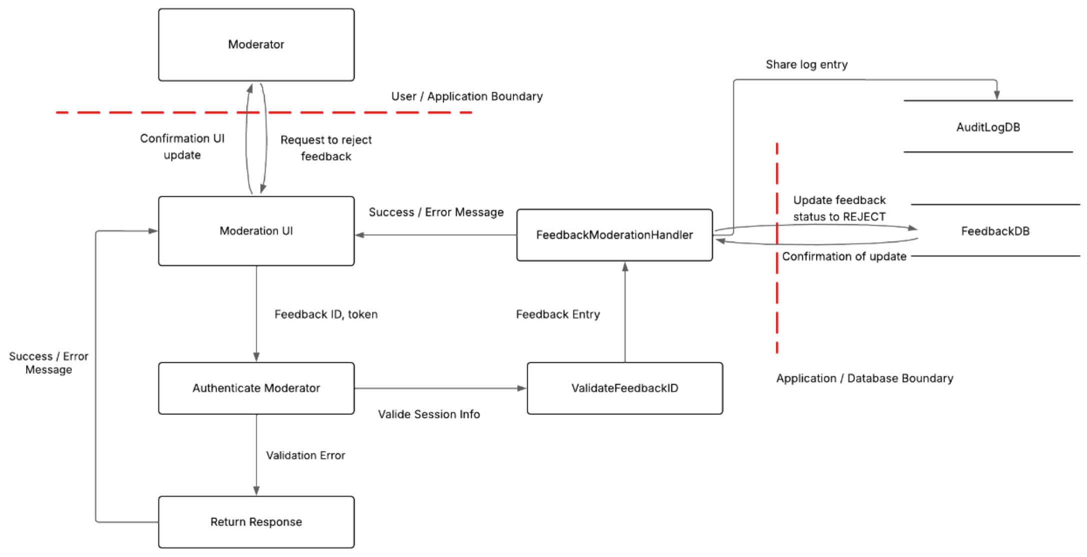


Figure 8 - DFD "Moderator reproves feedback"

5.7 Event Manager alters an event

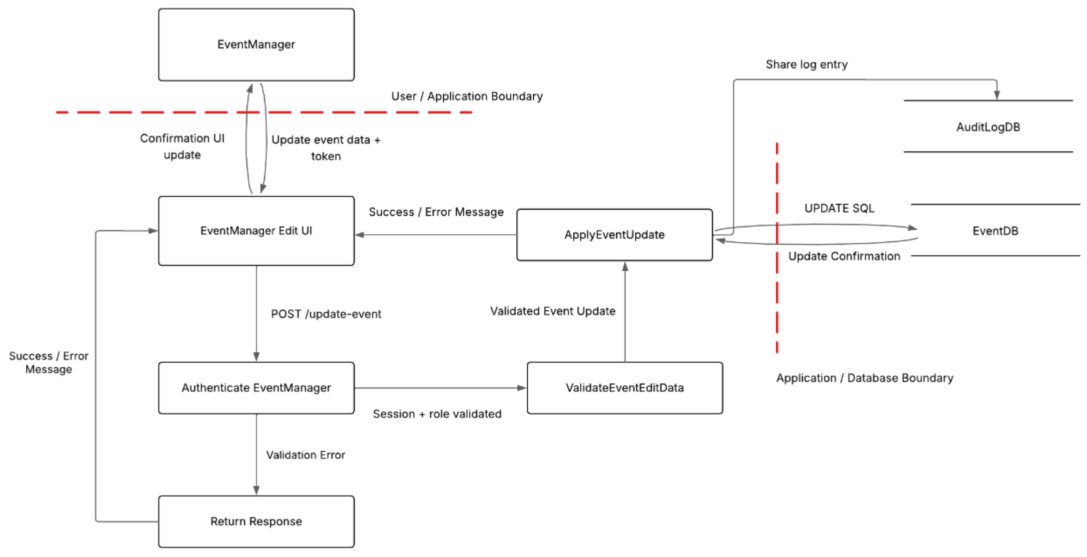


Figure 9 - DFD "Event Manager alters an event"

5.8 User filters events by booking date

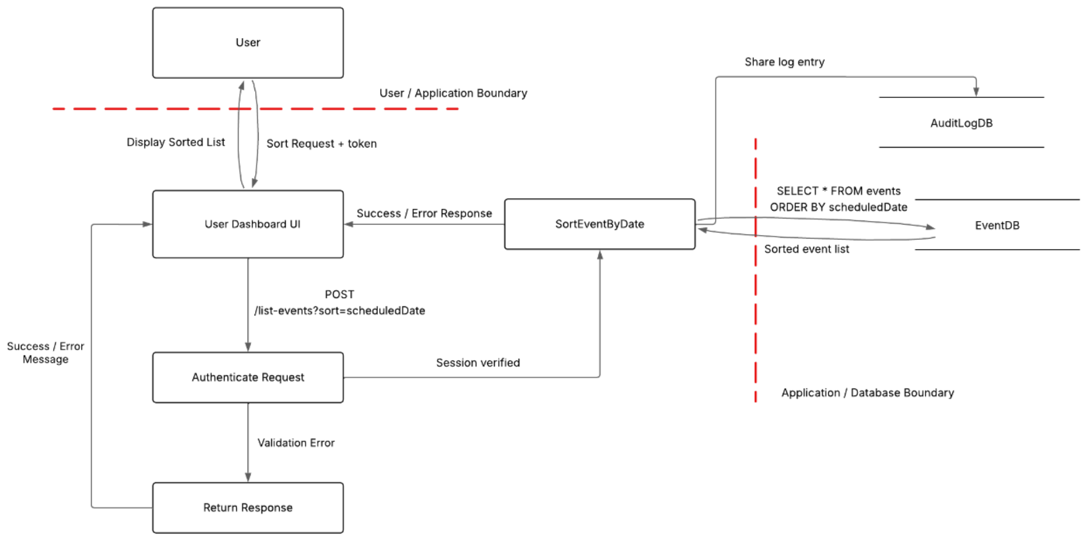


Figure 10 - DFD "User filters events by booking date"

5.9 Admin creates a category for events

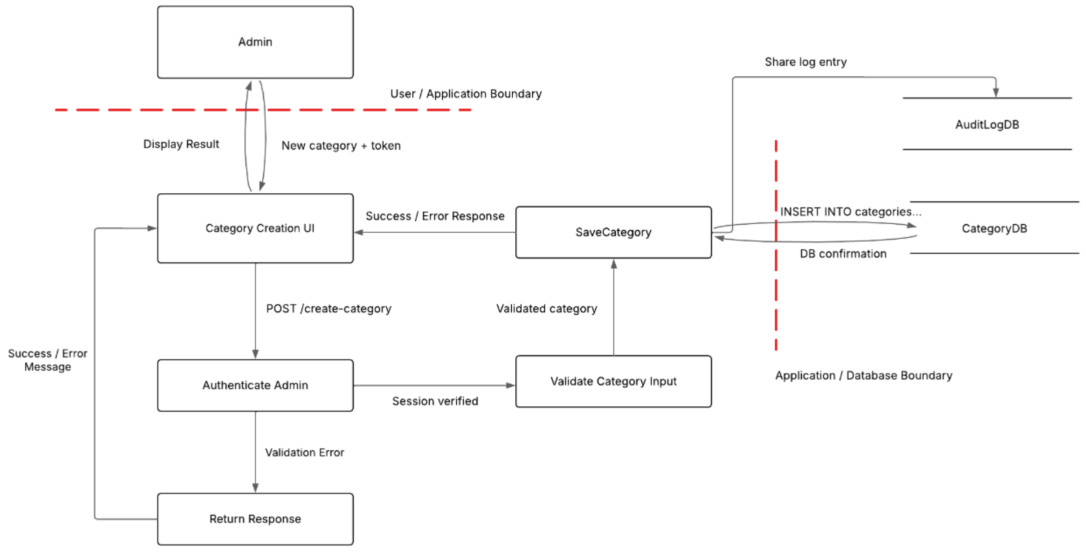


Figure 11 - DFD "Admin creates a category for events"

5.10 Client registers on a event

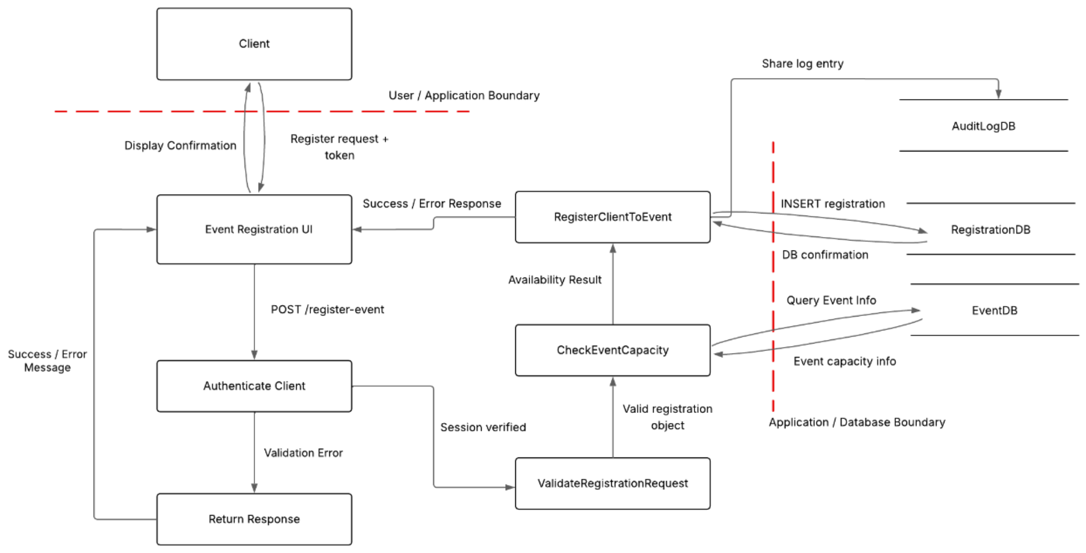


Figure 12 - DFD "Client registers on a event"

6 Tools

Tools to Use:

- **Version Control**: Git (GitHub or GitLab for collaboration)
- **CI/CD**: Jenkins, GitLab CI, or GitHub Actions
- **SAST/DAST Tools**: SonarQube, OWASP ZAP
- **Security Libraries**: JWT for secure authentication, bcrypt for password hashing
- **Database**: Use a relational database like MySQL or PostgreSQL
- **Developing Language**: Python
- **Deploy**: StreamLit

7 Domain-Driven Design

How DDD Should Be Designed for an Event Management System:

For an **event management system**, DDD can help structure the application around key concepts that represent the core functionality of the blog. Below are the essential steps and components to consider in designing the system using DDD principles:

Core Domains and Subdomains:

Core Domain

The core domain of your platform is **Event Management**. This is the primary focus of your application, involving creation, management, and participation in events.

Subdomains

- **User Management**: Manages user registration, authentication, and role-based access control.
- **Feedback Management**: Handles the collection, submission, and processing of feedback related to events.
- **Supplier Management**: Manages the suppliers/partners offering services for events (e.g., catering, security).
- **Moderation**: Involves moderating event submissions and user-generated content (comments/feedback).
- **Customer Support**: Handles user issues, inquiries, and complaints.

Aggregates:

In DDD, an **aggregate** is a cluster of domain objects that can be treated as a single unit. Aggregates ensure consistency and maintain the integrity of your domain.

For the tech blog, you might define the following aggregates:

- **User Aggregate**: Represents the different users of the system (e.g., Admin, Event Manager, Client/Participant, Moderator, Supplier/Partner). This aggregate handles authentication and authorization logic.
- **Event Aggregate**: This is the core of the system, as it manages event details and handles business logic like registration and attendee management.
- **Feedback Aggregate**: This handles the evaluation of events, storing ratings and comments submitted by users.

Bounded Contexts:

In DDD, a **Bounded Context** is a conceptual boundary within which a specific domain model is defined and applicable. For your project, the following bounded contexts could be defined:

- **Event Management Context**:

This context handles all aspects of event creation, modification, registration, and feedback.

- **User Management Context**:

This context handles user creation, role management, and authentication.

- **Feedback Management Context**:

This context handles the submission, retrieval, and management of event feedback.

- **Supplier Management Context**:

This context handles the management of suppliers and their services for events.

Each of these contexts can have its own database schema and logic, separated from other contexts.

Implement the Relationships:

- **Event ↔ User**:
    - One Event can have multiple Users as participants.
    - **Relationship**: One-to-many (One Event to many Users).
- **Event ↔ Feedback**:
    - One Event can have multiple Feedback entries.
    - **Relationship**: One-to-many (One Event to many Feedbacks).
- **User ↔ Feedback**:
    - A User can submit feedback for multiple events.
    - **Relationship**: One-to-many (One User to many Feedbacks).
- **Event ↔ Supplier**:
    - A Supplier can provide services for multiple Events.
    - **Relationship**: Many-to-many (Many Suppliers to many Events).

Define Security and Roles (Authorization):

Roles define access levels to various parts of the platform. This is crucial for ensuring that each user can only perform actions appropriate to their role.

- **Roles**:
    - **Administrator**: Full access to the system (manage users, events, and view statistics).
    - **Event Manager**: Can create, edit, cancel events, and manage event registration and participants.
    - **Client/Participant**: Can register for events, provide feedback, and view events they are registered for.
    - **Supplier/Partner**: Can add, edit, and remove services provided for events.

    - **Customer Support**: Can respond to user queries and complaints, monitor user experience.

- **Role-Based Access Control (RBAC)**:
    - Implement role-based permissions to ensure users can only access features specific to their role.
    - Example: Only **Admin** can manage users, and only **Event Manager** can create or cancel events.
- **Authentication and Authorization**:
    - Use **JWT** (JSON Web Tokens) or **OAuth2** for secure user authentication and authorization across the platform.

Design the Data Layer (Database Structure):

The data layer consists of the database schema and how data is structured. Here's how you can design it:

- **Entities and Tables**:
    - **Users**: id, username, password_hash, email, role_id (FK).
    - **Events**: id, name, description, date, location, created_by (FK).
    - **Feedback**: id, user_id (FK), event_id (FK), rating, comment, submitted_at.
    - **Suppliers**: id, company_name, services_offered, contact_info.
- **Relationships in the Database**:
    - **Users to Roles**: One-to-many (User has one Role).
    - **Events to Users**: Many-to-many (Users register for multiple Events).
    - **Events to Feedback**: One-to-many (Event can have many Feedback entries).
    - **Suppliers to Events**: Many-to-many (Supplier offers services to multiple Events).

Design API Endpoints:

The API is the interface through which the frontend will interact with the backend. Below are some key API endpoints.

- **User API**:
    - POST /api/register – Registers a new user.
    - POST /api/login – Authenticates a user.
    - GET /api/users/{id} – Retrieves user profile.
    - PUT /api/users/{id} – Updates user profile.
- **Event API**:
    - POST /api/events – Creates a new event (Event Manager).

    - GET /api/events/{id} – Gets details of a specific event.
    - PUT /api/events/{id} – Updates event details (Event Manager).
    - DELETE /api/events/{id} – Cancels the event (Event Manager).
- **Feedback API**:
    - POST /api/events/{id}/feedback – Submits feedback for an event.
    - GET /api/events/{id}/feedback – Retrieves all feedback for an event.
    - PUT /api/feedback/{id} – Updates feedback.
- **Supplier API**:
    - POST /api/suppliers – Adds a new supplier.
    - GET /api/suppliers/{id} – Retrieves supplier details.
    - PUT /api/suppliers/{id} – Updates supplier services.

Threat Modeling:

Threat modeling is a proactive approach to identifying potential vulnerabilities in the system and designing mitigations. Use the **STRIDE** model to analyze threats.

- **Spoofing**: Prevent unauthorized users from pretending to be someone else. Use **multi-factor authentication**.
- **Tampering**: Prevent unauthorized modification of data. Use **data encryption** and **integrity checks**.
- **Repudiation**: Ensure actions are properly logged so users can't deny actions. Use **audit logs** for all sensitive actions.
- **Information Disclosure**: Protect sensitive user information. Use **encryption** and **secure transmission protocols (HTTPS)**.
- **Denial of Service (DoS)**: Ensure the platform can handle high loads without crashing. Use **load balancing** and **rate limiting**.
- **Elevation of Privilege**: Ensure users can't gain higher privileges than their assigned role. Implement **RBAC** and **secure session management**.

Security Considerations:

Security is a top priority in your system, particularly around **data protection**, **user authentication**, and **access control**. Key security measures include:

- **Authentication**: Use **JWT tokens** for stateless authentication, ensuring that users only have access to allowed endpoints.
- **Authorization**: Implement **role-based access control (RBAC)** so that only authorized users can perform certain actions.

- **Encryption**: Ensure that sensitive data (passwords, personal details) are **encrypted** both in transit (using **SSL/TLS**) and at rest.
- **Input Validation**: Perform **input validation** and **sanitization** to avoid injection attacks like **SQL injection** and **XSS**.

8 STRIDE

8.1 Spoofing (Identity Fraud)

Description: Spoofing occurs when an attacker pretends to be another legitimate user (Admin, Event Manager, Client, etc.) to gain unauthorized access or perform malicious actions.

Potential Threats:

- Account Impersonation: An attacker impersonates Admin, Event Manager, or Client to perform actions like creating/canceling events or accessing sensitive data.
- Session Hijacking: An attacker steals an authenticated session (e.g., JWT) and gains control of an active user session.
- OAuth Token Theft: Stolen OAuth2 tokens allow attackers to bypass authentication mechanisms.
- API Key Theft (for external integrations): Attackers misuse leaked API keys to act as trusted systems.

Mitigation Strategies:

- Implement Multi-Factor Authentication (MFA) for Admins, Event Managers, and Moderators.
- Use JWT tokens with short expiration times and automatic renewal mechanisms.
- Set Secure, HttpOnly, and SameSite attributes on cookies.
- Enable OAuth2 token revocation upon logout or inactivity.
- Apply Device Fingerprinting to detect anomalies during login.
- Integrate CAPTCHA to prevent automated login attacks.
- Enforce Mutual TLS (mTLS) for communication with trusted external services.

8.2 Tampering (Data Manipulation)

Description: Tampering involves the unauthorized modification of data, such as event details, feedback comments, or user roles.

Potential Threats:

- Event Data Manipulation: Changing event information (date, location) maliciously.
- Feedback Tampering: Editing or submitting fake/inappropriate feedback to distort event reputation.
- Supplier Service Alteration: Manipulating supplier details without authorization.
- Man-in-the-Middle (MitM) Attacks: Intercepting and altering communication between client and server.

Mitigation Strategies:

- Implement strict RBAC (Role-Based Access Control) to enforce who can modify specific resources.
- Apply Input Validation and Output Encoding to prevent client-side tampering.
- Use Hashing or Digital Signatures for sensitive data verification.
- Use TLS 1.3 for secure communications at all stages.
- Activate Content Security Policy (CSP) to prevent client-side script manipulation.

8.3 Repudiation (Denying Actions)

Description: Repudiation involves users denying actions like event cancellation or feedback submission, often without verifiable records.

Potential Threats:

- Event Deletion Denial: Event Managers deny canceling or editing events.
- Feedback Submission Denial: Clients deny submitting abusive comments.
- Role Assignment Denial: Admins deny assigning incorrect roles.

Mitigation Strategies:

- Enable Full Audit Logging with timestamped, tamper-evident records.
- Use Immutable Logging Storage (append-only) for all critical operations.
- Implement Digital Signatures for sensitive actions (event creation, feedback approval).
- Regularly audit logs and correlate actions with user IP and device information.

8.4 Information Disclosure (Data Exposure)

Description: Unauthorized exposure of sensitive platform information, including personal data, feedback, event statistics, and supplier profiles.

Potential Threats:

- Sensitive Event Information Leakage: Unauthorized users accessing hidden event details.
- User Credential Exposure: Passwords, emails, or payment data being leaked.
- Verbose Error Messages: Revealing stack traces or system configurations.
- Third-party Data Breach: Through insecure APIs for payments/events.

Mitigation Strategies:

- Encrypt all sensitive data at rest (AES-256) and in transit (TLS).
- Apply Strict RBAC policies for access to sensitive data.
- Implement Data Masking/Redaction techniques for non-admin users.
- Replace detailed error messages with generic user-facing errors, while logging the full stack trace internally.
- Regularly review third-party service compliance (e.g., PCI DSS for payments).

8.5 Denial of Service (DoS)

Description: Attacks that attempt to make the platform unavailable by overloading resources.

Potential Threats:

- Event Registration Flood: Bots mass-register for events to cause database overload.
- Feedback Spam Attack: Submitting thousands of fake feedback entries.
- Resource Exhaustion: Uploading oversized files or complex queries.
- Algorithmic Complexity Attacks: Exploiting computationally expensive API endpoints.

Mitigation Strategies:

- Apply Rate Limiting on all user actions, especially registration and feedback APIs.
- Use WAF (Web Application Firewall) to detect and block DoS patterns.
- Implement CAPTCHA on registration and feedback forms.
- Ensure Auto-scaling infrastructure to absorb legitimate spikes.
- Limit upload sizes and validate input data thoroughly.

8.6 Elevation of Privilege (Escalating Permissions)

Description: Elevation of privilege occurs when users gain unauthorized rights and access to restricted functionalities.

Potential Threats:

- Client Becomes Admin: Exploiting API vulnerabilities to assign themselves a higher role.
- Supplier Access to Event Management: Gaining permissions to create or edit events.
- Broken Access Controls: Unprotected APIs allowing sensitive actions without validation.

Mitigation Strategies:

- Implement Strict Server-side RBAC with role validation at each endpoint.
- Use Attribute-Based Access Control (ABAC) for fine-grained permissions.
- Validate user role at each critical action on the server (never trust frontend validations).
- Perform periodic Permission Reviews to remove unnecessary privileges.
- Secure APIs with OAuth2 Scopes and JWT Claims Validation.
- Perform Pentesting and Authorization Testing regularly.

9 Threat Ranking

9.1 Overview

The threat ranking for the Event Management Platform was carried out using the DREAD model. This method considers five factors: Damage Potential, Reproducibility, Exploitability, Affected Users, and Discoverability. Each factor is rated on a scale of 1 (low) to 10 (high). The average determines the overall threat level.

9.2 Threat Tables

9.2.1 Unauthorized Role Escalation

Metric	Value
Damage	9
Reproducibility	7
Exploitability	8
Affected Users	8
Discoverability	7
Average Score	7.8
Risk Level	High

Table 11 - Unauthorized Role Escalation

9.2.2Token Theft (JWT or OAuth)

Metric	Value
Damage	9
Reproducibility	7
Exploitability	8
Affected Users	9
Discoverability	8
Average Score	8.2
Risk Level	High

Table 12 - Token Theft (JWT or OAuth)

9.2.3 Event Registration DoS Attack

Metric	Value
Damage	7
Reproducibility	8
Exploitability	9
Affected Users	8
Discoverability	9
Average Score	8.2
Risk Level	High

Table 13 - Event Registration DoS Attack

9.2.4 SQL Injection (Login/Feedback)

Metric	Value
Damage	8
Reproducibility	7
Exploitability	7
Affected Users	7
Discoverability	7
Average Score	7.2
Risk Level	High

Table 14 - SQL Injection (Login/Feedback)

9.2.5 Feedback Data Exposure

Metric	Value
Damage	8
Reproducibility	6
Exploitability	6
Affected Users	7
Discoverability	7
Average Score	6.8
Risk Level	Medium

Table 15 - Feedback Data Exposure

9.2.6 Supplier Service Tampering

Metric	Value
Damage	7
Reproducibility	6
Exploitability	6
Affected Users	6
Discoverability	6
Average Score	6.2
Risk Level	Medium

Table 16 - Supplier Service Tampering

9.2.7 Lack of Audit Trail

Metric	Value
Damage	6
Reproducibility	5
Exploitability	5
Affected Users	5
Discoverability	5
Average Score	5.2
Risk Level	Medium

Table 17 - Lack of Audit Trail

9.2.8 Inappropriate Feedback (not moderated)

Metric	Value
Damage	6
Reproducibility	5
Exploitability	4
Affected Users	5
Discoverability	5
Average Score	5.0
Risk Level	Low

Table 18 - Inappropriate Feedback (not moderated)

9.2.9 Resource Exhaustion Uploads

Metric	Value
Damage	6
Reproducibility	6
Exploitability	7
Affected Users	6
Discoverability	6
Average Score	6.2
Risk Level	Medium

Table 19 - Resource Exhaustion Uploads

9.2.10 Verbose Error Messages (Information Disclosure)

Metric	Value
Damage	7
Reproducibility	5
Exploitability	5
Affected Users	6
Discoverability	6
Average Score	5.8
Risk Level	Medium

Table 20 - Verbose Error Messages (Information Disclosure)

9.3 Explanation

- Unauthorized Role Escalation and Token Theft are the highest risks due to the critical nature of unauthorized access.
- DoS attacks during event registration are highly reproducible and affect platform availability.
- SQL Injection remains a major risk if input validation is bypassed.
- Feedback Data Exposure, Supplier Service Tampering, and Resource Exhaustion represent serious but more controlled risks.
- Audit trail weaknesses and inappropriate feedback are still concerns but with less direct damage.

10 Abuse cases

To ensure the security and reliability of the EventFlow platform, it is important not only to define how the system should behave under normal conditions, but also to anticipate how it could be misused. Abuse cases help us identify potential vulnerabilities by exploring malicious actions that could compromise the platform's functionality, data integrity, or user trust. By analyzing these scenarios, we are able to design better protections, improve user roles and permissions, and strengthen the overall resilience of the system against real-world threats.

10.1 Event Manager creates an Event

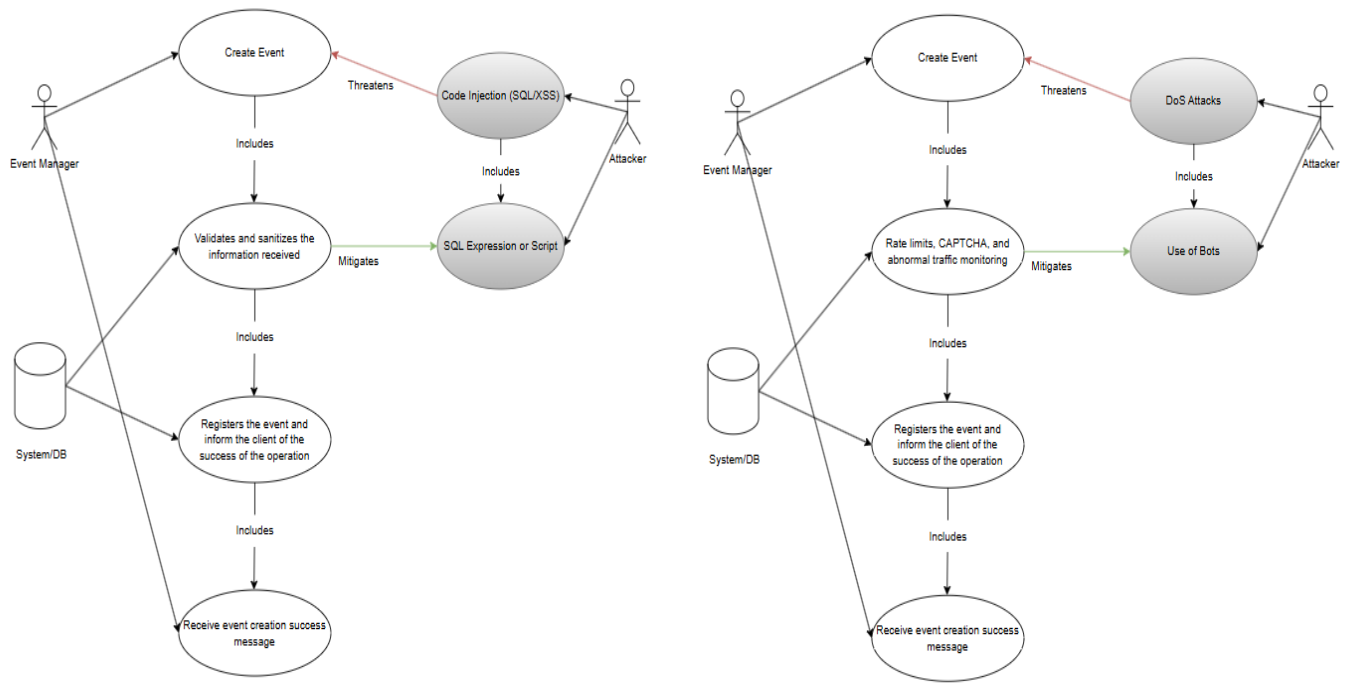

Create Event
Threatens
Code Injection (SQL/XSS)
Event Manager
Includes
Includes
Attacker
Validates and sanitizes the information received
Mitigates
SQL Expression or Script
Includes
System/DB
Registers the event and inform the client of the success of the operation
Includes
Receive event creation success message
Create Event
Threatens
DoS Attacks
Event Manager
Includes
Includes
Attacker
Rate limits, CAPTCHA, and abnormal traffic monitoring
Mitigates
Use of Bots
Includes
System/DB
Registers the event and inform the client of the success of the operation
Includes
Receive event creation success message

Figure 13 - Abuse Case "Event Manager creates an Event"

10.2 Client gives feedback to an event

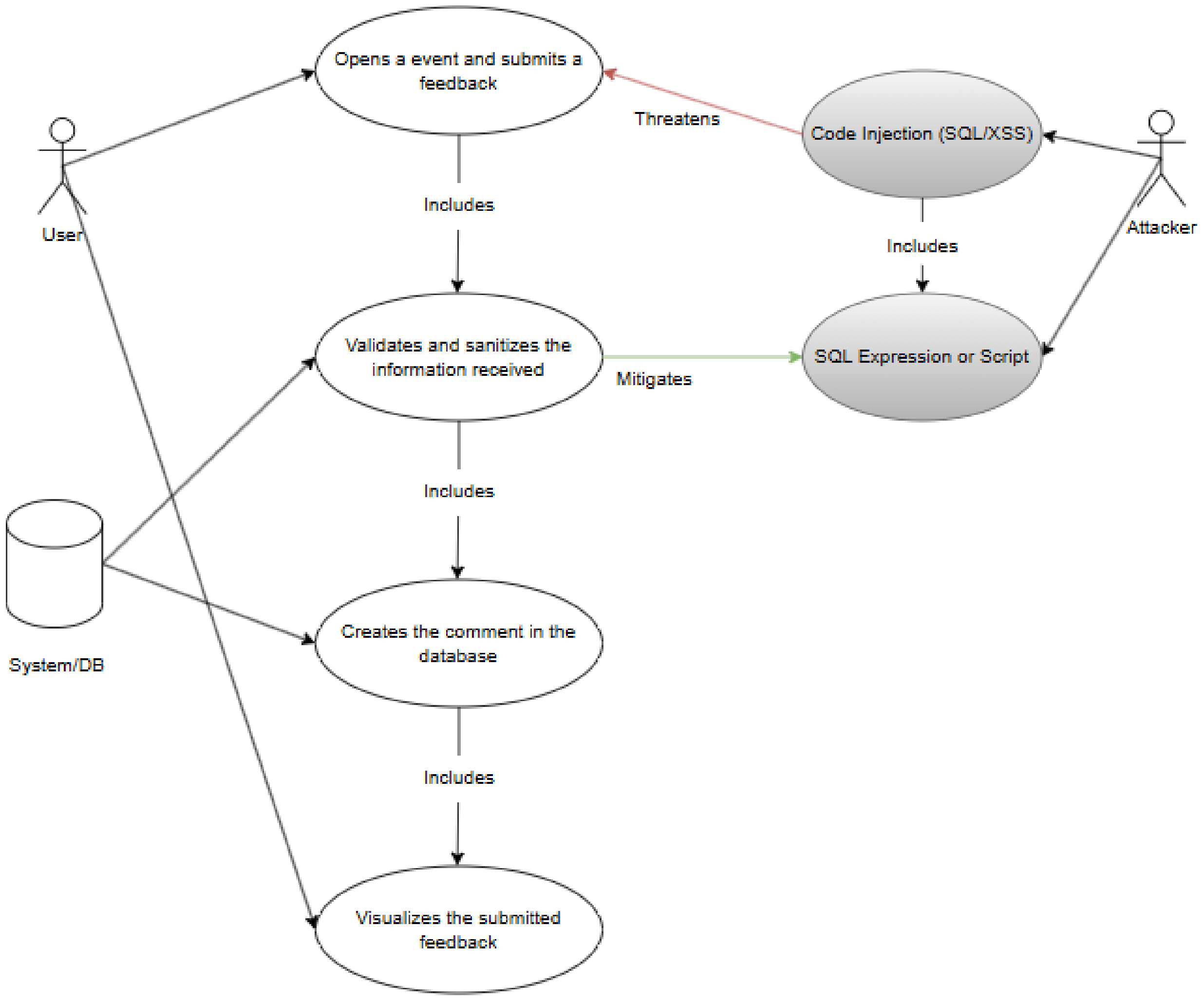

Opens a event and submits a feedback
Threatens
Code Injection (SQL/XSS)
Attacker
User
Includes
Includes
Validates and sanitizes the information received
Mitigates
SQL Expression or Script
Includes
System/DB
Creates the comment in the database
Includes
Visualizes the submitted feedback

Figure 14 - Abuse Case "Client gives feedback to an event"

10.3 Admin creates user and attributes a role

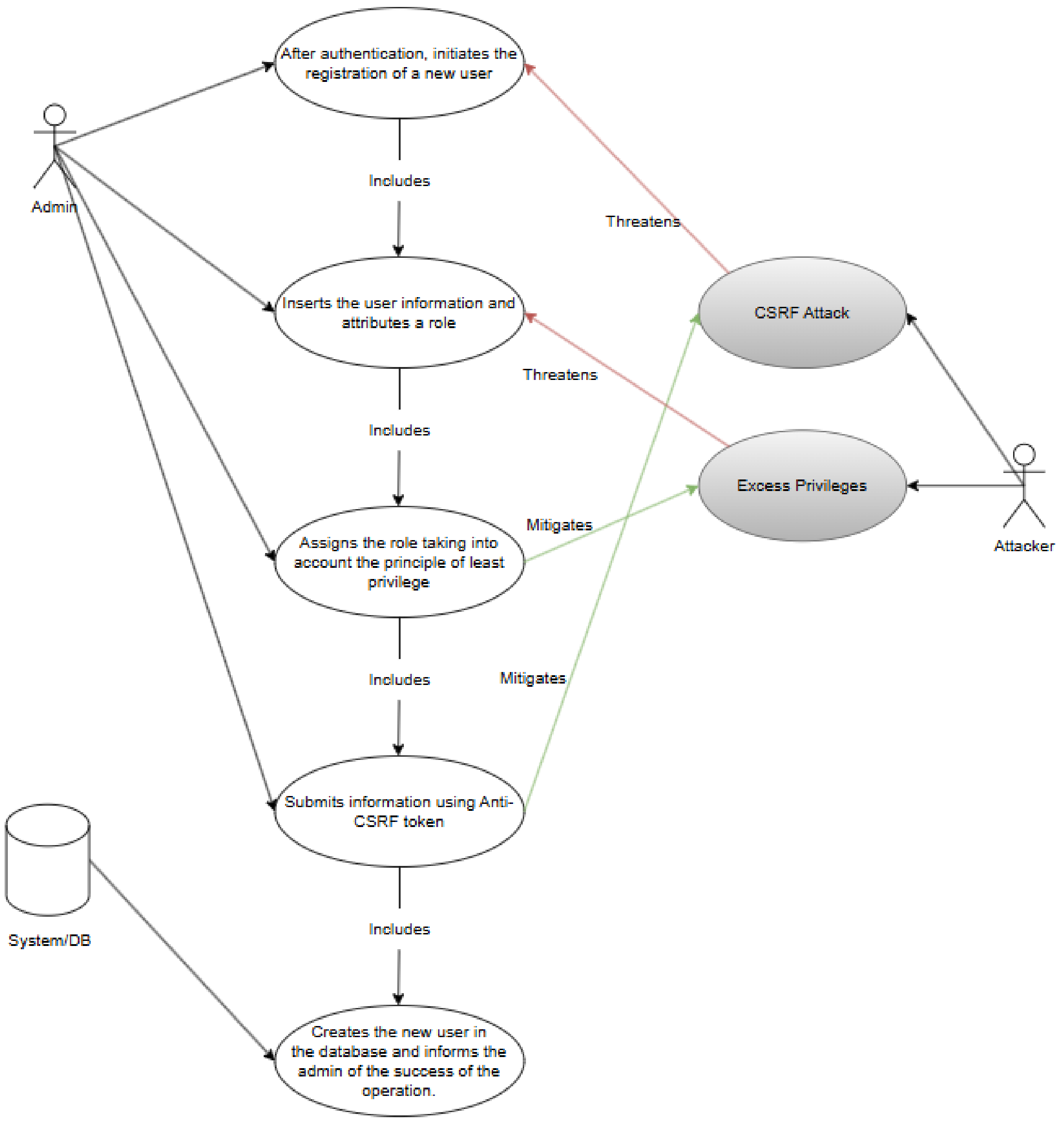

After authentication, initiates the registration of a new user
Admin
Includes
Threatens
Inserts the user information and attributes a role
CSRF Attack
Threatens
Includes
Excess Privileges
Mitigates
Assigns the role taking into account the principle of least privilege
Attacker
Includes
Mitigates
Submits information using Anti-CSRF token
System/DB
Includes
Creates the new user in the database and informs the admin of the success of the operation.

Figure 15 - Abuse Case "Admin creates user and attributes a role"

10.4 Client logins with credentials

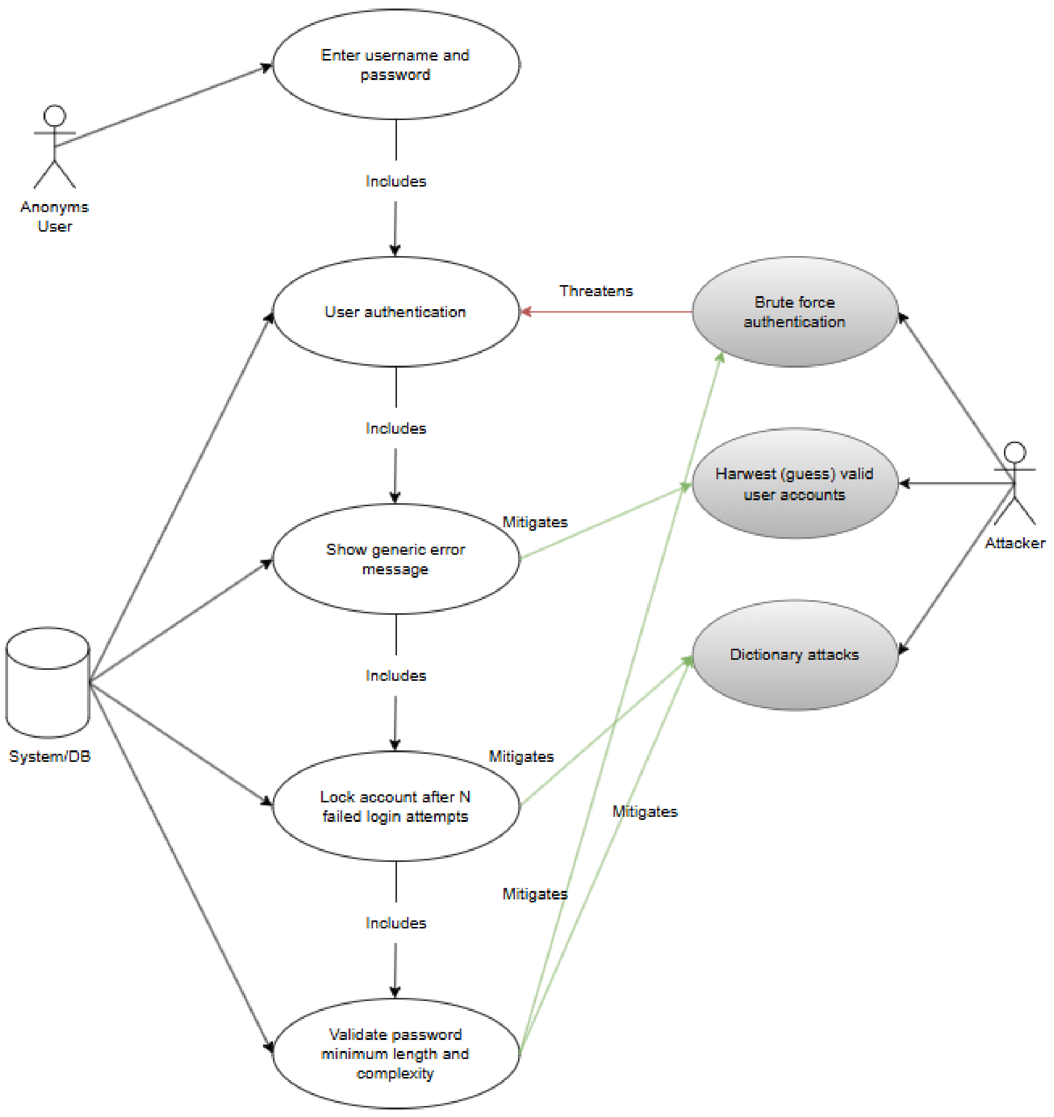
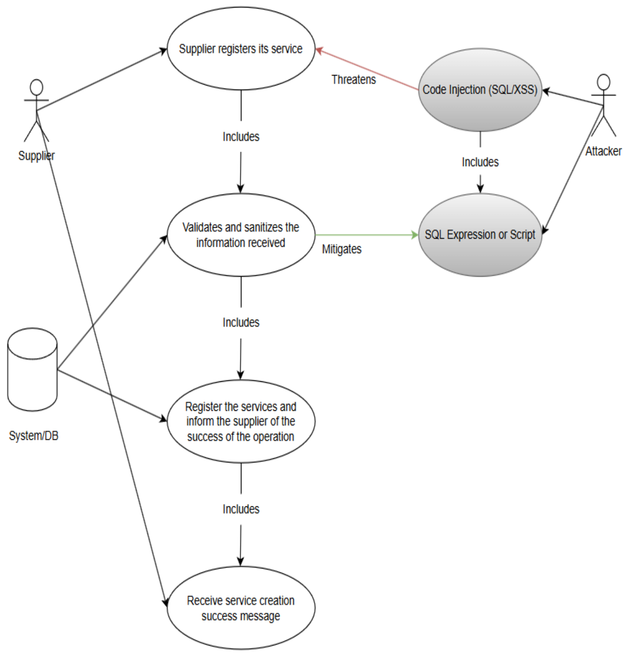

Enter username and password
Anonyms User
Includes
User authentication
Threatens
Brute force authentication
Includes
Harwest (guess) valid user accounts
Attacker
Mitigates
Show generic error message
Includes
Dictionary attacks
System/DB
Mitigates
Lock account after N failed login attempts
Mitigates
Mitigates
Includes
Validate password minimum length and complexity

Figure 16 - Abuse Case "Client logins with credentials"

10.5 Supplier registers his services


Supplier registers its service
Threatens
Code Injection (SQL/XSS)
Supplier
Includes
Includes
Attacker
Validates and sanitizes the information received
Mitigates
SQL Expression or Script
Includes
System/DB
Register the services and inform the supplier of the success of the operation
Includes
Receive service creation success message

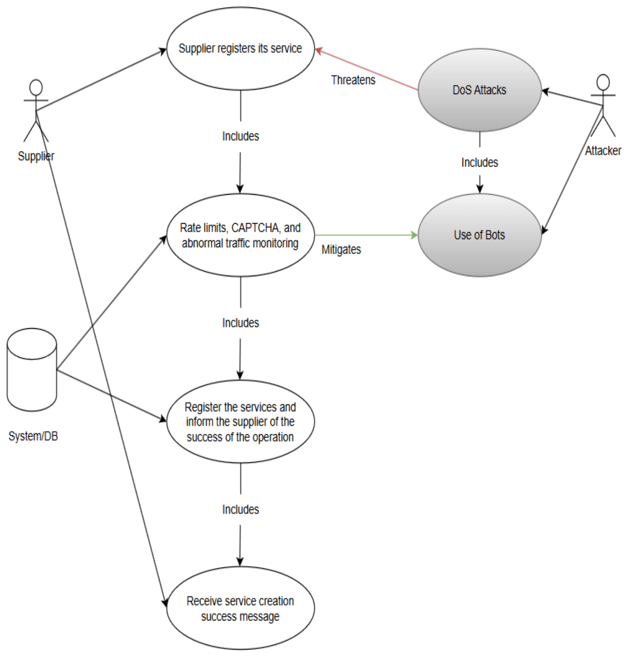

Supplier registers its service
Threatens
DoS Attacks
Supplier
Includes
Includes
Attacker
Rate limits, CAPTCHA, and abnormal traffic monitoring
Mitigates
Use of Bots
Includes
System/DB
Register the services and inform the supplier of the success of the operation
Includes
Receive service creation success message

Figure 17 - Abuse Case "Supplier registers his services"

10.6 Event Manager alters an event

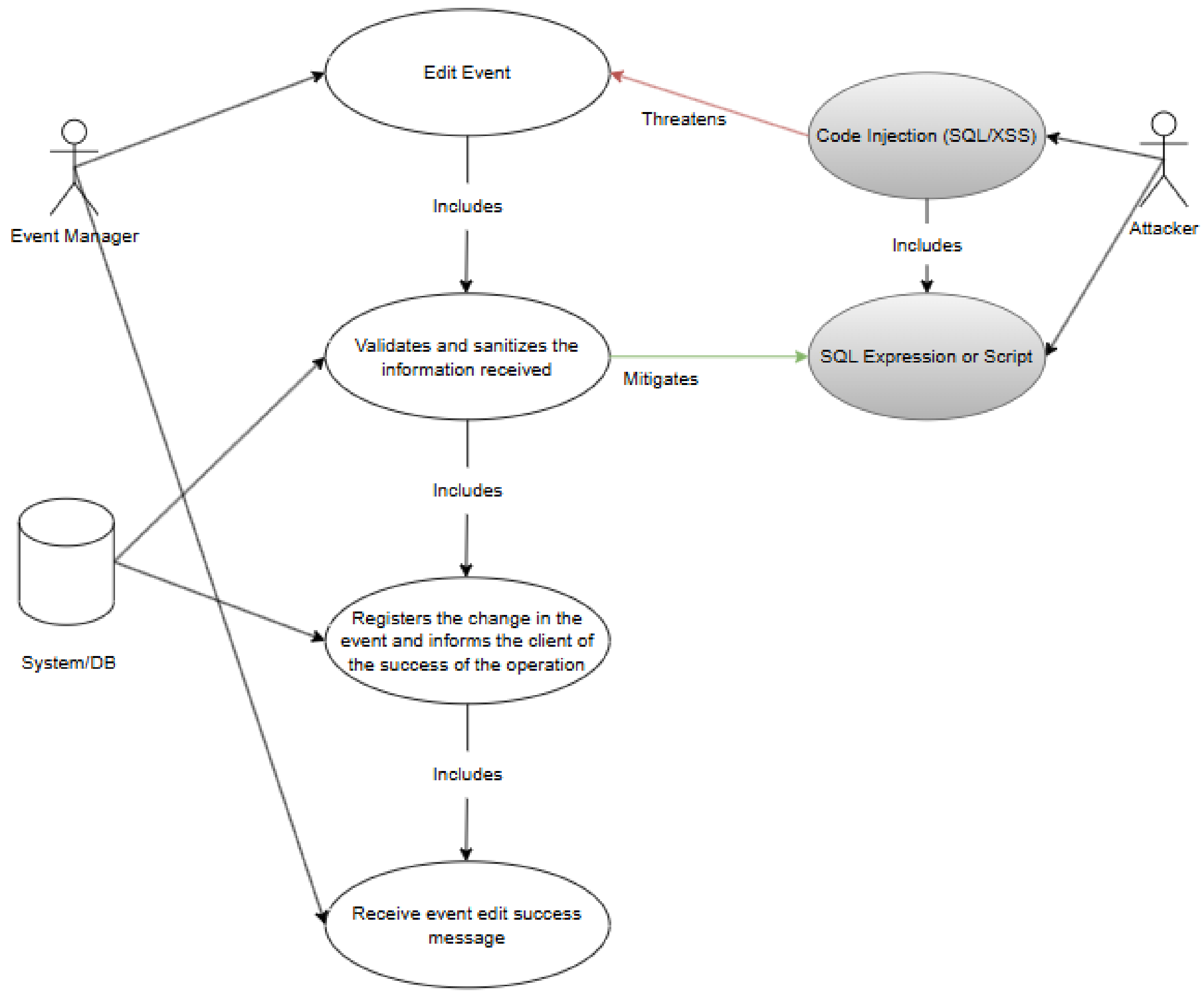
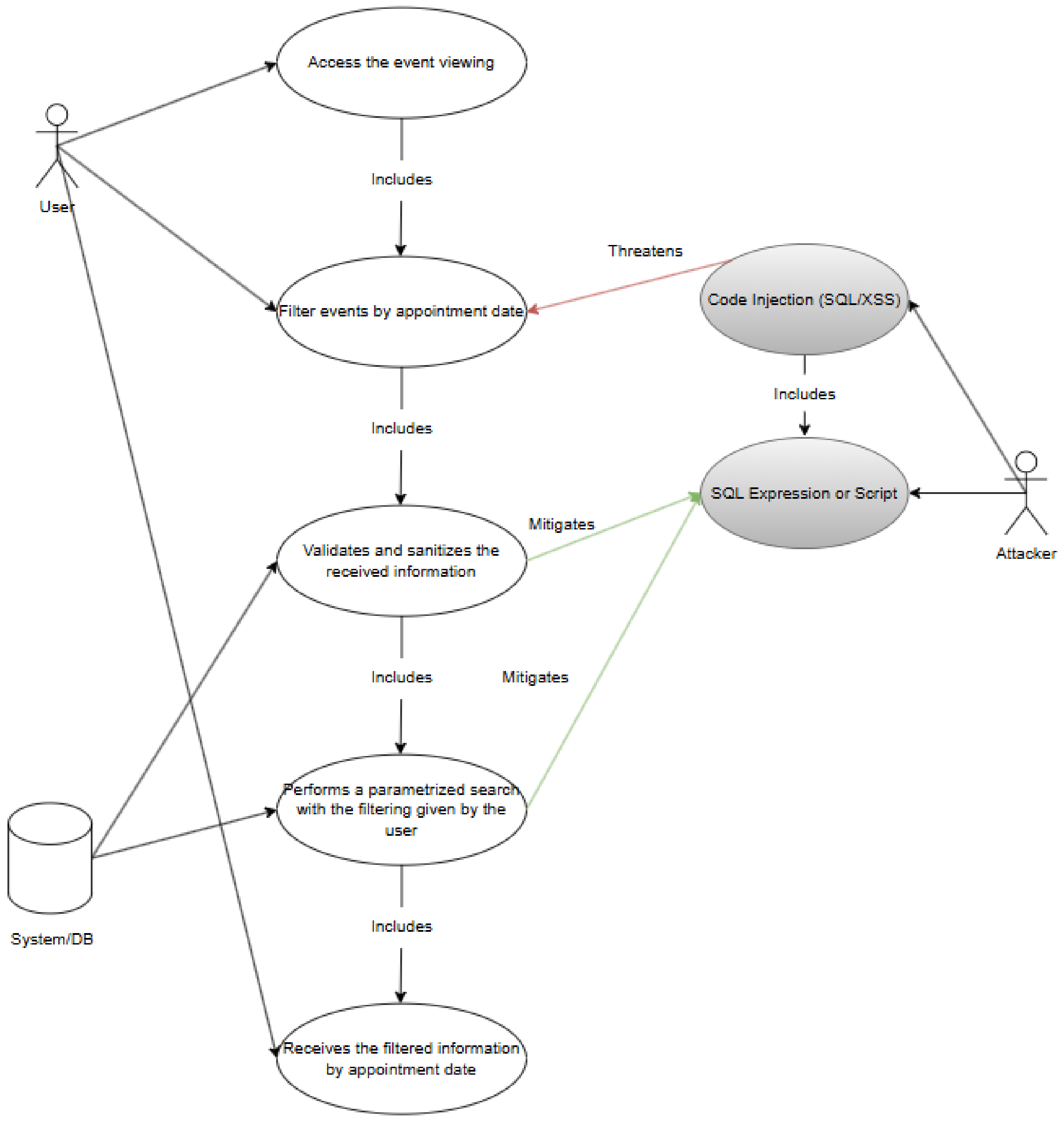

Edit Event
Event Manager
Includes
Threatens
Code Injection (SQL/XSS)
Attacker
Includes
Validates and sanitizes the information received
Mitigates
SQL Expression or Script
System/DB
Includes
Registers the change in the event and informs the client of the success of the operation
Includes
Receive event edit success message

Figure 18 - Abuse Case "Event Manager alters an event"

10.7 User filters events by booking date



Access the event viewing
Includes
User
Threatens
Filter events by appointment date
Code Injection (SQL/XSS)
Includes
Includes
SQL Expression or Script
Mitigates
Attacker
Validates and sanitizes the received information
Includes
Mitigates
Performs a parametrized search with the filtering given by the user
System/DB
Includes
Receives the filtered information by appointment date

Figure 19 - A buse Case "User filters events by booking date"

10.8 Admin creates a category for events

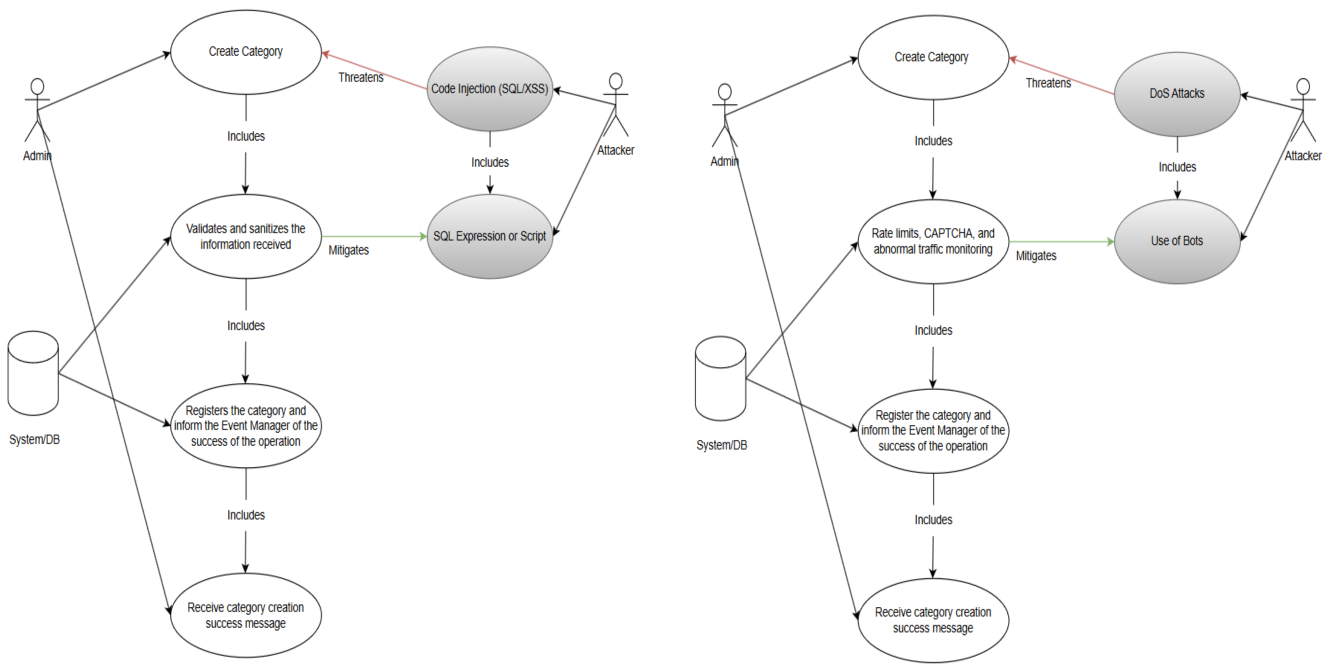

Create Category
Threatens
Code Injection (SQL/XSS)
Admin
Includes
Includes
Attacker
Validates and sanitizes the information received
Mitigates
SQL Expression or Script
Includes
System/DB
Registers the category and inform the Event Manager of the success of the operation
Includes
Receive category creation success message
Create Category
Threatens
DoS Attacks
Admin
Includes
Includes
Attacker
Rate limits, CAPTCHA, and abnormal traffic monitoring
Mitigates
Use of Bots
Includes
System/DB
Register the category and inform the Event Manager of the success of the operation
Includes
Receive category creation success message

Figure 20 - Abuse Case "Admin creates a category for events"

10.9 Client registers on a event

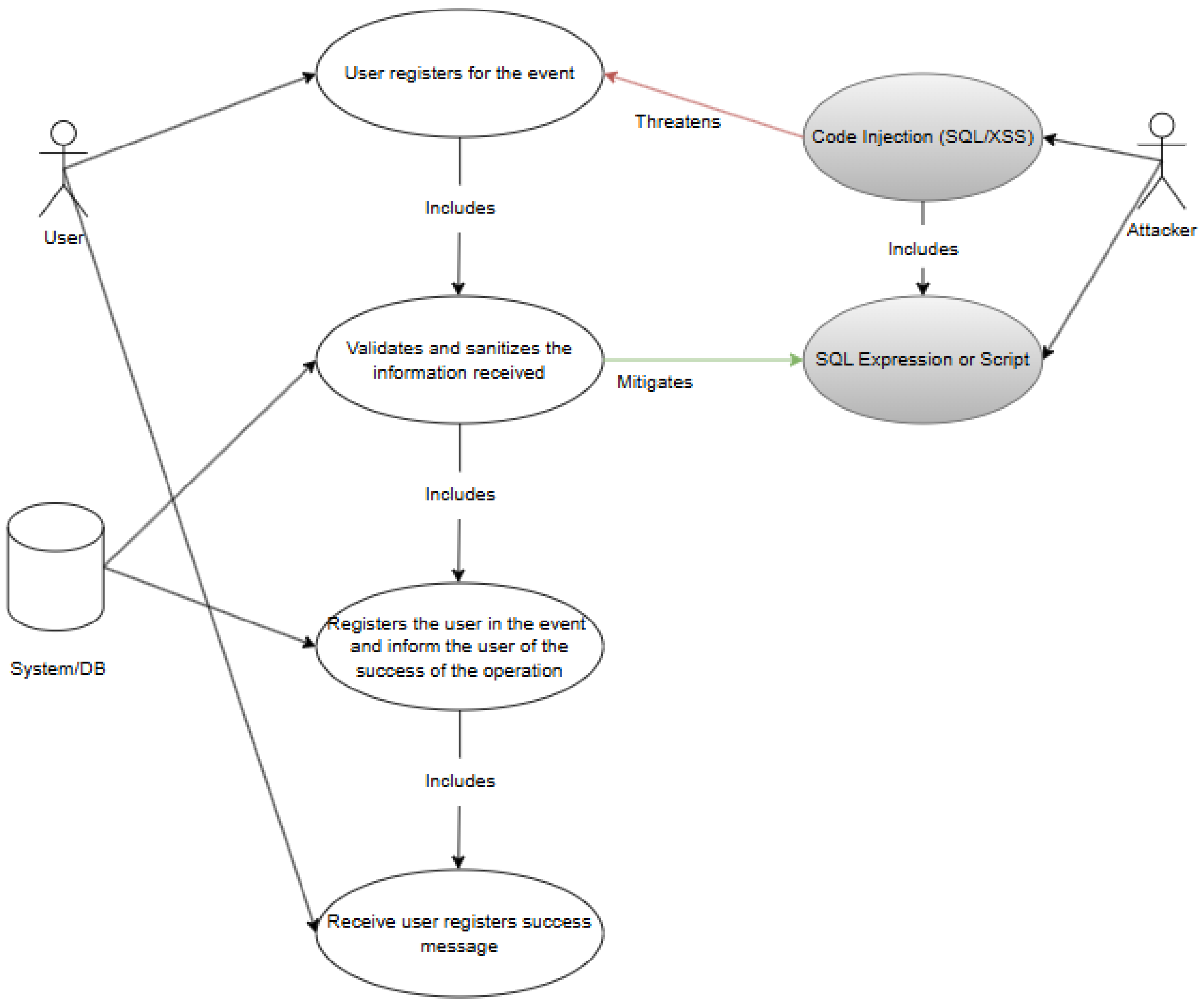
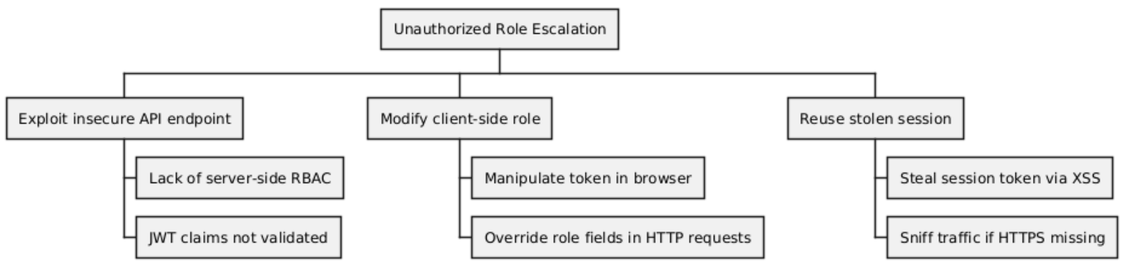

User registers for the event
Threatens
Code Injection (SQL/XSS)
User
Includes
Includes
Attacker
Validates and sanitizes the information received
Mitigates
SQL Expression or Script
Includes
System/DB
Registers the user in the event and inform the user of the success of the operation
Includes
Receive user registers success message

Figure 21-Abuse Case "Client registers on a event"

11 Attack Trees

In this section, we present Attack Trees for the main threats identified in the STRIDE and Threat Ranking of the Event Management Platform. An Attack Tree models the possible steps an attacker could take to achieve a malicious goal. Each tree starts with the main objective at the top and branches into different actions or conditions that an attacker might exploit. The following diagrams use UML (Unified Modeling Language) to represent the attack paths in a structured and hierarchical way.

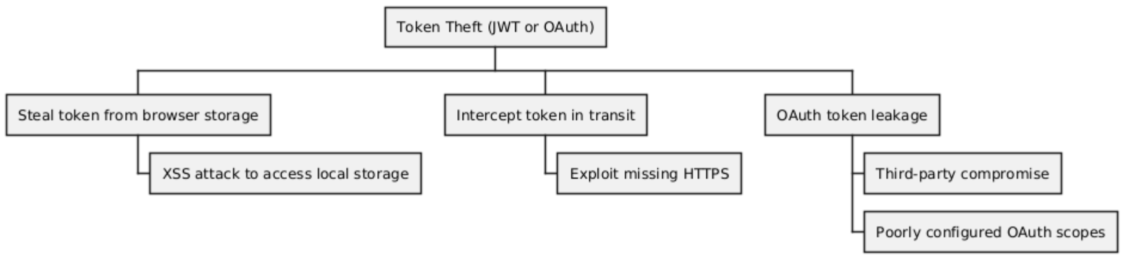
Unauthorized Role Escalation



Unauthorized Role Escalation
Exploit insecure API endpoint
Lack of server-side RBAC
JWT claims not validated
Modify client-side role
Manipulate token in browser
Override role fields in HTTP requests
Reuse stolen session
Steal session token via XSS
Sniff traffic if HTTPS missing

Figure 22 - Unauthorized Role Escalation

Token Theft



Token Theft (JWT or OAuth)
Steal token from browser storage
XSS attack to access local storage
Intercept token in transit
Exploit missing HTTPS
OAuth token leakage
Third-party compromise
Poorly configured OAuth scopes

Figure 23 - Token Theft

SQL Injection

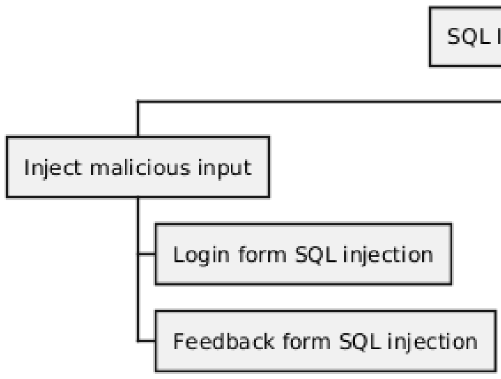


Figure 24 - SQL Injection

Denial Of Service

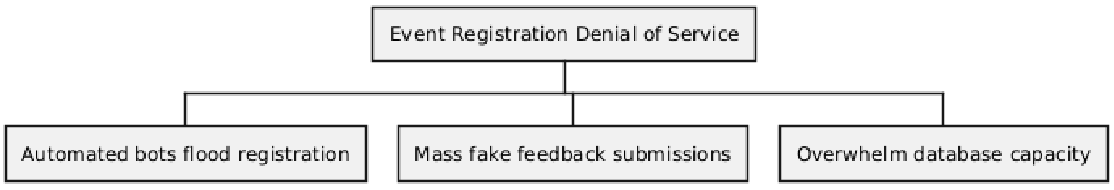

Event Registration Denial of Service
Automated bots flood registration
Mass fake feedback submissions
Overwhelm database capacity

Figure 25 - Denial Of Service

Information Disclosure

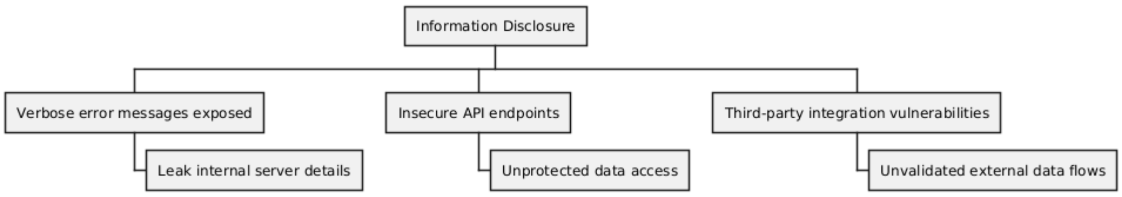

Information Disclosure
Verbose error messages exposed
Insecure API endpoints
Third-party integration vulnerabilities
Leak internal server details
Unprotected data access
Unvalidated external data flows

Figure 26 - Information Disclosure

12 Qualitative Risk Model

Identify, categorize, and prioritize **security, operational, and functional risks** using a qualitative scale based on:

- **Likelihood**: How likely the risk is to occur (Low, Medium, High)
- **Impact**: The severity if the risk occurs (Low, Medium, High)
- **Risk Level**: Combined judgment (e.g., High Impact + Medium Likelihood = High Risk)

ID	Risk Description	Category	Likelihood	Impact	Risk Level	Mitigation
R1	Unauthorized role escalation	Security	Medium	High	High	Strict RBCA, server-side validation, audit logs
R2	Event registration DoS (bot floods)	Security/ Availability	High	Medium	High	CAPTCHA, rate limiting, WAF
R3	SQL Injection on login or feedback forms	Security	Medium	High	High	ORM usage, input sanitization, WAF
R4	Inappropriate feedback not moderated	Functional	Medium	Medium	Medium	Add moderation queue, role: Moderator
R5	Supplier offers duplicated/conflict services	Business Logic	Low	Medium	Low	Unique service constraints, admin validation
R6	Feedback data exposure (unauthenticated users can view all)	Security/ Confidentiality	Medium	High	High	Enforce user/session checks before retrieval
R7	Token theft (JWT replay or session hijacking)	Security	Medium	High	High	Short token lifespan, token refresh, HTTPS-only, HtppOnly flags

R8	Failure to assign roles properly during registration	Functional	Medium	Medium	Medium	Role assigned via secure admin flows only
R9	Poor scalability with 1000s of concurrent user	Performance	Low	High	Medium	Enable horizontal scaling and load balancing
R10	Lack of audit trail for changes	Compliance	Medium	Medium	Medium	Add audit logging across all major actions

13 Conclusion

Throughout this project, we set out to design and document the EventFlow platform with a clear focus on building something secure, user-friendly, and scalable. We followed a structured approach, combining best practices like Domain-Driven Design, threat modeling with STRIDE and DREAD, and careful definition of both functional and non-functional requirements to create a solid foundation for the system.

Security was a key priority from the start, with strong authentication, role-based access control, and data protection measures integrated into the platform. At the same time, we made sure the system would be easy and intuitive for users to navigate, while also being reliable and efficient behind the scenes.

By analyzing risks and planning effective countermeasures, we were able to anticipate potential issues and strengthen the platform against them. Thanks to the modular design and clearly defined domains, EventFlow is ready to grow and adapt in the future, making it easier to add new features without losing control over security or performance.

In short, this project lays down a strong, thoughtful foundation for EventFlow a platform designed not just to work well today, but to keep evolving and meeting new challenges with confidence.