Workshop No. 2 — Design Artifacts and System Modeling

Equipo Alfa Buena Maravilla Onda Dinamita Escuadrón Lobo

Juan David Buitrago Salazar – jbuitragosa@unal.edu.co

Luis David Garzón Morales – lgarzonmo@unal.edu.co

Juan David Cárdenas Galvis – jcardenasgal@unal.edu.co

Álvaro Camilo Torres Rodriguez – alvtorresro@unal.edu.co

David Felipe Chaparro Perez – dachaparrop@unal.edu.co

Professor

Liliana Marcela Olarte Mesa

Universidad Nacional de Colombia

Department of Systems and Industrial Engineering

Software Engineering II

Bogotá D.C., Colombia

2025

1 CRC Cards

CRC Cards allows us to see in a first instance the way our designed system is going to work based on the Object Oriented Programming paradigm, defining the systems in classes like entities with characteristics (attributes) and actions (methods) including interactions between these to establish the business logic for our CTF platform. **The next CRC cards show main classes definitions with their responsibilities and collaborations.**

1.1 User

User (Abstract class)	
Responsibilities:	**Collaborations:**
• Manage user authentication and credentials. • Monitor account status and security. • Handle password changes and resets. • Track login attempts and timestamps. • Provide a basis for all user types.	• Administrator (inherits from User) • Participant (inherits from User) • PasswordResetRequest (for recovery) • UserService (for user operations)

1.2 Administrator

Administrator	
Responsibilities:	**Collaborations:**
• Create, edit, and delete challenges • Configure event parameters • Reset user passwords • View system statistics • Manage challenge categories	• Challenge (manage challenges) • Event (configure events) • Category (manage categories) • EventService (use for operations) • ChallengeService (use for challenges)

1.3 Participant

Participant	
Responsibilities:	**Collaborations:**
• Join and leave teams • Send flags for challenges • View challenges and leaderboard • Manage individual scores • Filter and sort challenges	• Team (belongs to a team) • Challenge (solves challenges) • Submission (sends flags) • SubmissionService (used for submissions) • TeamService (used for teams)

1.4 Team

Team	
Responsibilities:	**Collaborations:**
• Manage team membership and roles • Control access with passwords • Track scores and challenges completed • Validate size limits • Identify team captain	• Participant (has members) • Event (validates limits) • Score (tracks scoring) • Submission (makes submissions) • TeamService (used by service)

1.5 Challenge

Class: Challenge	
Responsibilities:	**Collaborations:**
• Define configuration and metadata • Validate submitted flags • Manage scoring strategy • Control visibility and status • Track solution counter • Manage attachments	• Category (belongs to category) • Flag (has validation flag) • File (contains files) • IScoringStrategy (uses strategy) • Submission (receives submissions) • Event (belongs to event)

1.6 Event

Class: Event	
Responsibilities:	**Collaborations:**
• Manage global CTF configuration • Control event timing and status • Define competition rules and limits • Configure rate limits and team sizes • Validate submission permissions	• Challenge (contains challenges) • Team (has participating teams) • RulesVersion (has rule versions) • Notification (generates notifications) • EventService (managed by service)

1.7 Submission

Class: Submission	
Responsibilities:	**Collaborations:**
• Record flag submission attempts • Track validation results • Store submission metadata • Manage rate limiting tracking • Maintain audit trail	• Participant (sent by participant) • Challenge (sent to challenge) • Team (submitted by team) • Score (results in score if correct)

1.8 Score

Class: Score	
Responsibilities:	**Collaborations:**
• Record points awarded for correct submissions • Calculate team and individual totals • Support score reversal • Link scores to specific submissions • Maintain scoring consistency	• Submission (comes from submission) • Team (scores team) • Participant (scores participant) • Challenge (comes from challenge)

1.9 Category

Class: Category	
Responsibilities:	**Collaborations:**

• Organize challenges by type/theme • Enable/disable categories • Prevent deletion when there are challenges • Provide filtering and sorting	• Challenge (categorize challenges) • Administrator (managed by admin)

1.10 EventService

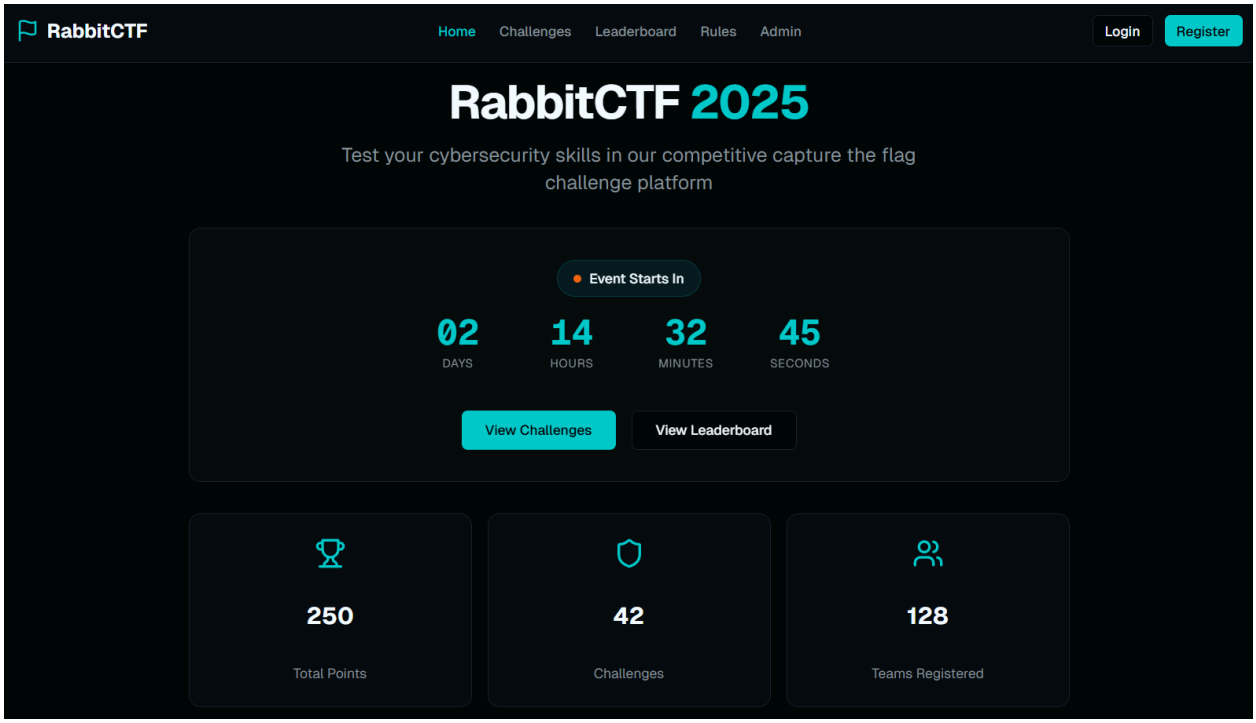
Class: EventService	
Responsibilities:	**Collaborations:**
• Control event lifecycle operations • Manage rule updates • Configure global settings and limits • Handle state transitions • Coordinate operations at the event level	• Event (manages events) • RulesVersion (manages versions) • NotificationService (triggers notifications)

2 Mockups

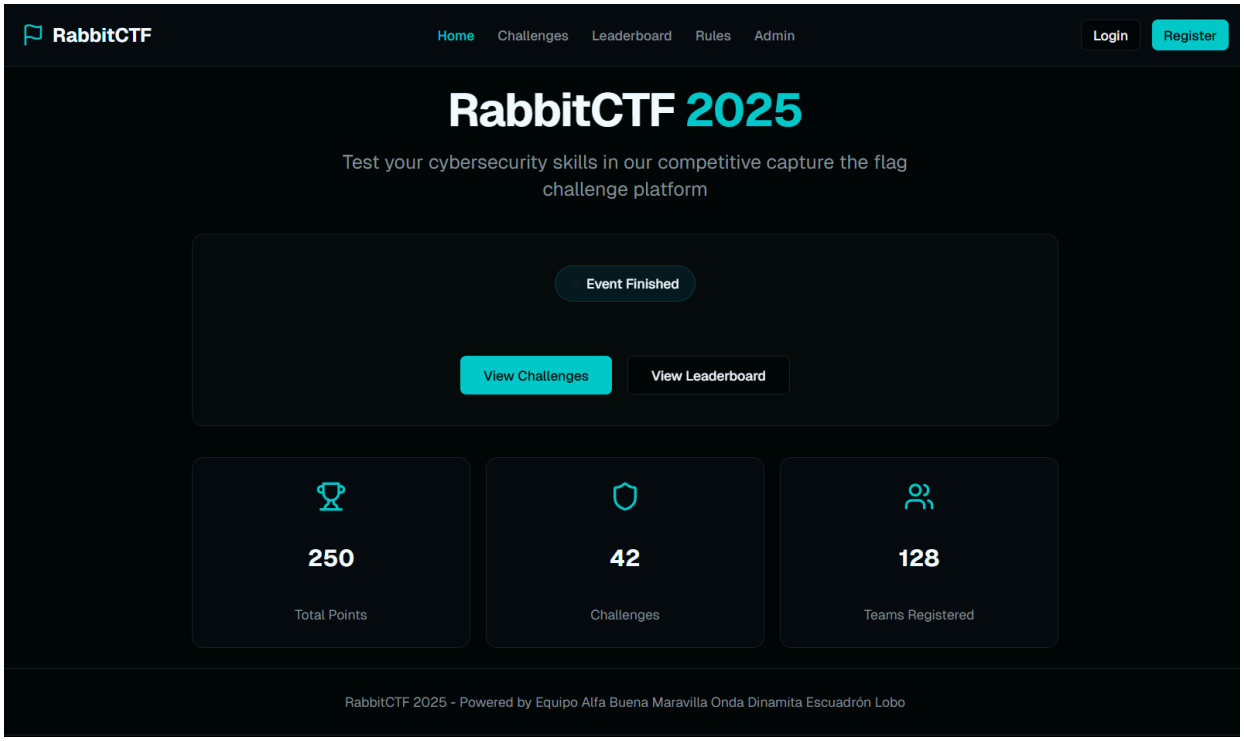
2.1 Home Page

The home page functions as the primary landing interface for the RabbitCTF platform, designed to provide an immediate and clear overview of the event's status and key information. Its layout is structured to guide the user's attention through a logical sequence, from a general introduction to specific actionable options and concluding with overarching event metrics.

The design features a consistent header with the event title, "RabbitCTF 2025," and a tagline explaining the platform's purpose. The central portion of the interface is dynamic, changing to reflect the current phase of the event. One state is configured for the period before the competition begins, prominently displaying a countdown timer that indicates the days, hours, minutes, and seconds until the event starts. This creates anticipation and informs users of the precise start time. Below this, two primary navigation buttons, "View Challenges" and "View Leaderboard," are offered, allowing users to preview the competition structure and rules.



A second, distinct state of the home page is presented for when the event has concluded. Here, the countdown is replaced by a definitive "Event Finished" banner, unambiguously communicating that the active competition period is over. The same navigation buttons remain accessible, enabling participants to retrospectively review the challenge set and consult the final, official leaderboard standings.

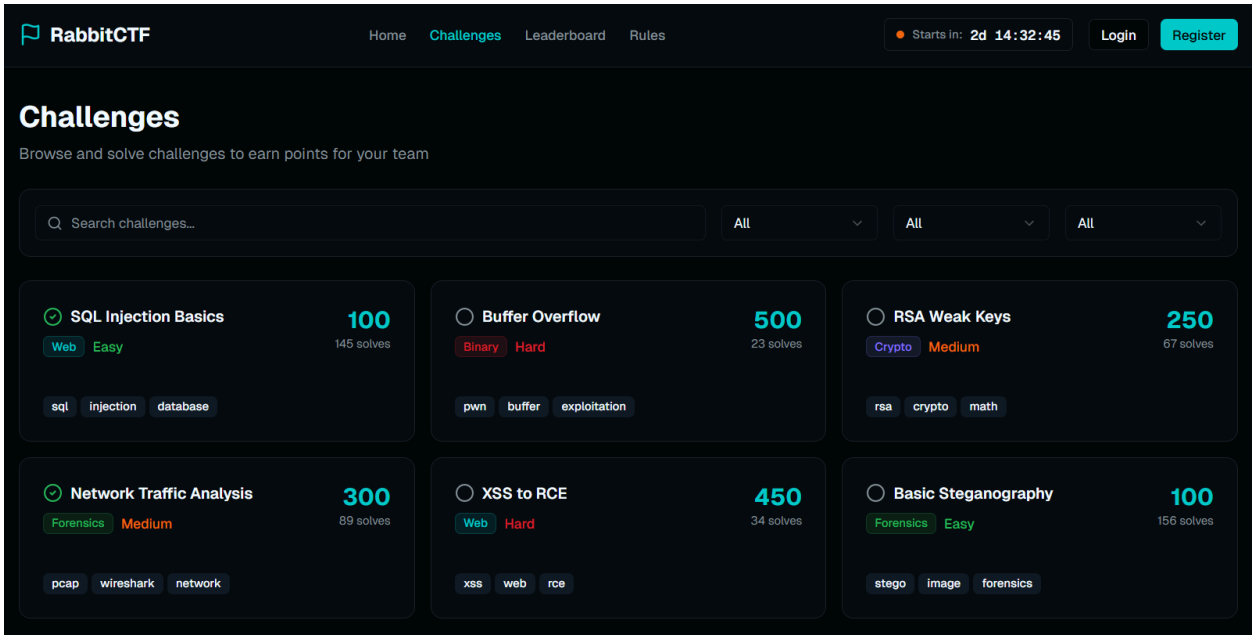


Both interface states conclude with an identical footer section that presents a summary of key event statistics: the total points available across all challenges, the total number of challenges, and the number of teams registered. This data provides a consistent snapshot of the event's scale. The page is finalized with a credit line identifying the development team, ensuring a professional and complete presentation.

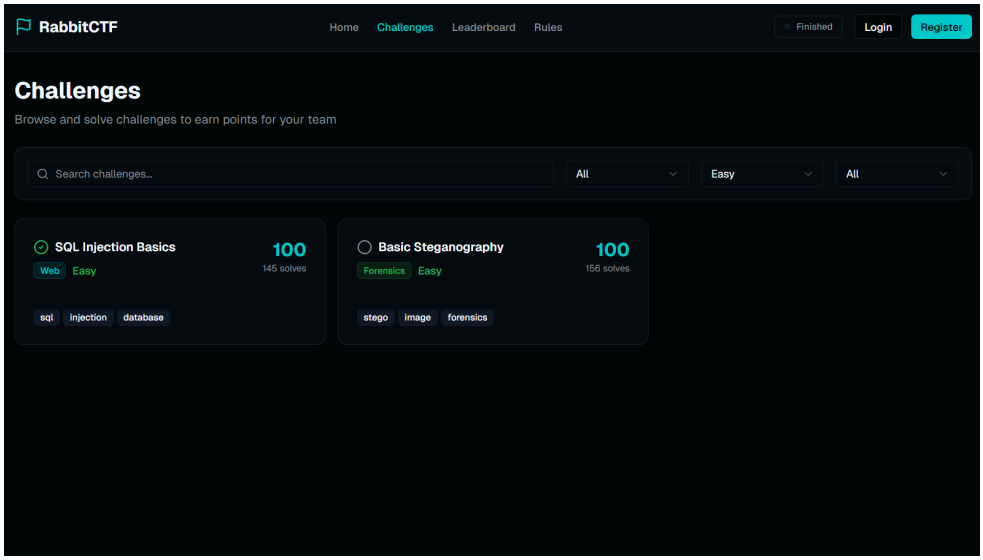
2.2 Challenges Page

The Challenges section is the core interactive area of the RabbitCTF platform, where participants browse, select, and attempt to solve the various cybersecurity problems. The interface is designed for efficiency and clarity, enabling users to quickly navigate a potentially large set of challenges and access the necessary details to begin working on them.

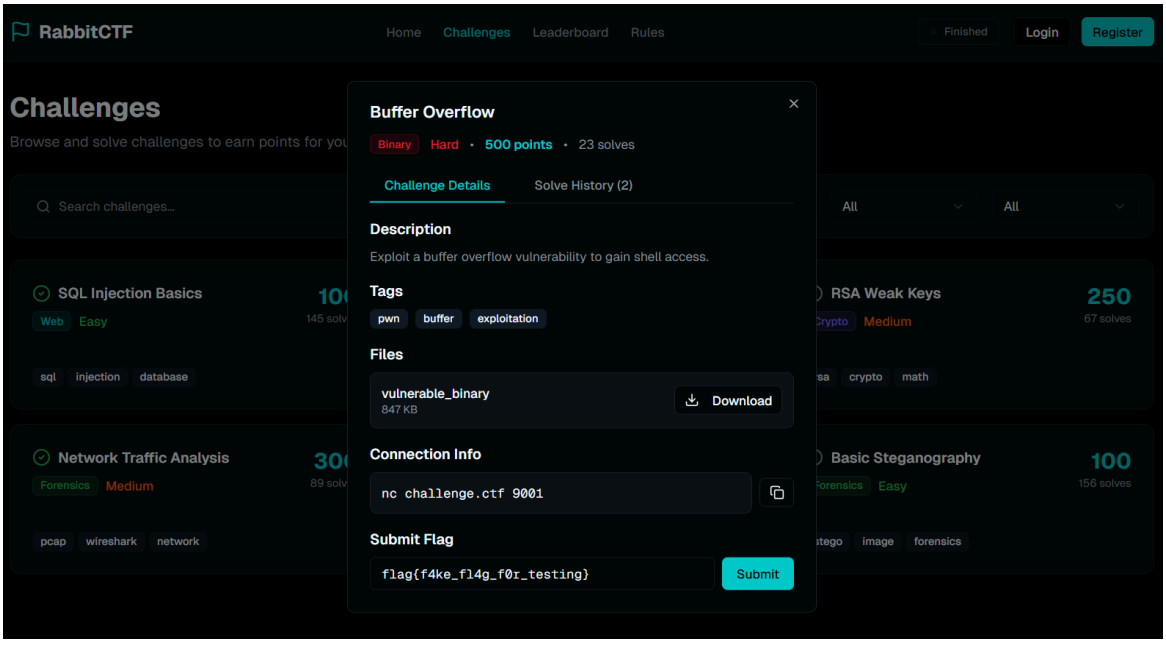
The primary view presents a comprehensive list of all available challenges. Each challenge is displayed as a compact card, consistently showing its title, category, difficulty level, point value, and the number of teams that have already solved it.



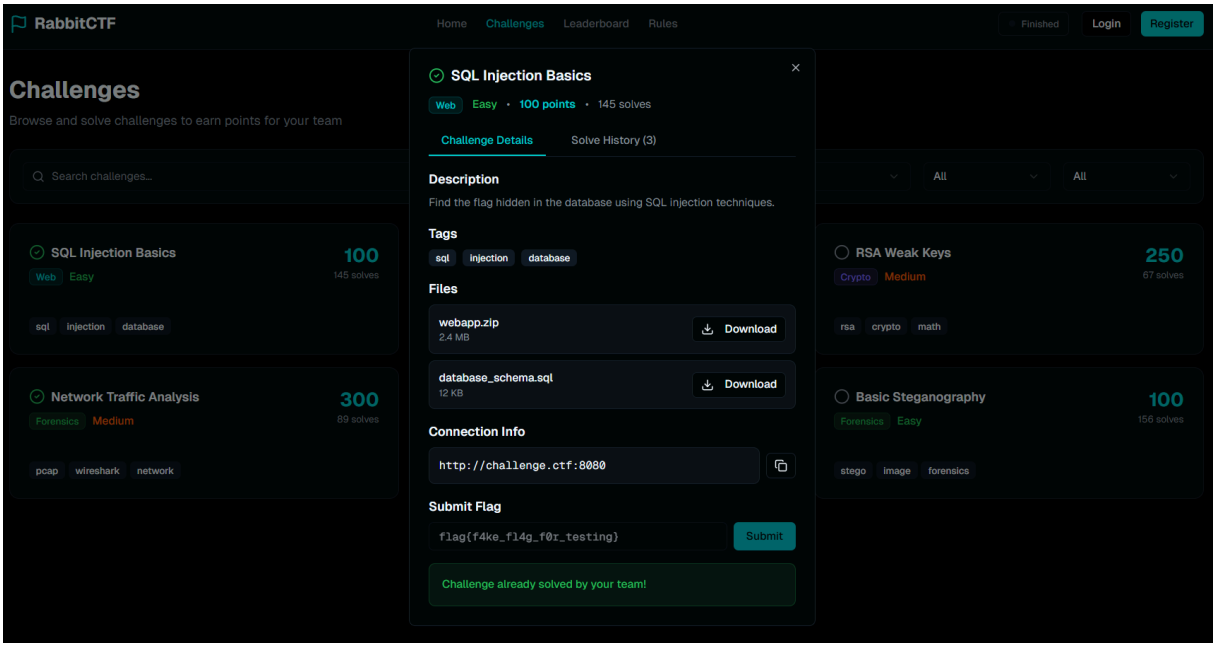
A search bar at the top allows for filtering by name, while a set of filter buttons enables sorting and narrowing of the list by criteria such as category and difficulty. This layout provides a high-level overview, allowing competitors to assess the field and identify targets based on their skills and the potential reward.



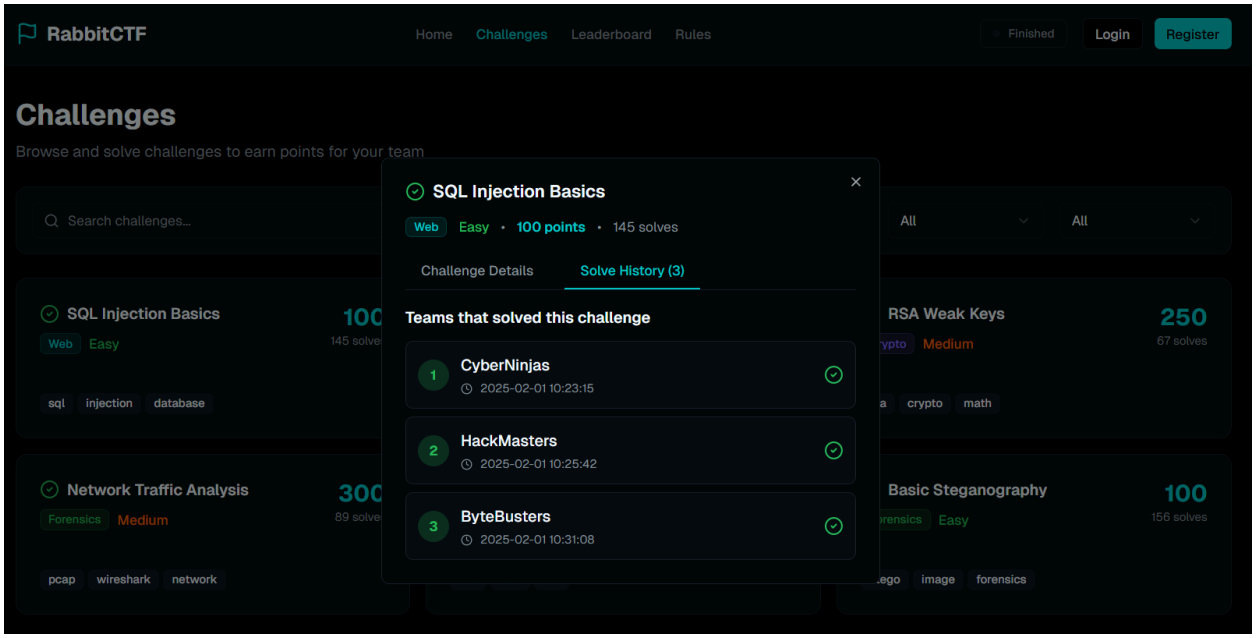
When a user selects a specific challenge, a detailed modal view is presented. This view is divided into several structured sections. The header consolidates the challenge's key attributes: its category, difficulty, point value, and solve count. A detailed description of the task follows, providing the context and goal. Associated tags offer further thematic classification. If the challenge requires external analysis, a "Files" subsection provides downloadable resources, while "Connection Info" supplies network addresses or commands to access a remote vulnerable service.



The central interactive element of the detailed view is the "Submit Flag" input field, where participants enter their proposed solutions. The interface provides immediate feedback; for instance, if a challenge has already been solved by the user's team, the input field is replaced by a notification confirming this status.



An adjacent "Solve History" section lists the teams that have successfully completed the challenge, fostering a sense of competition and allowing participants to track the progress of others. This comprehensive design ensures that all necessary tools and information are centralized, streamlining the problem-solving workflow for the competitor.

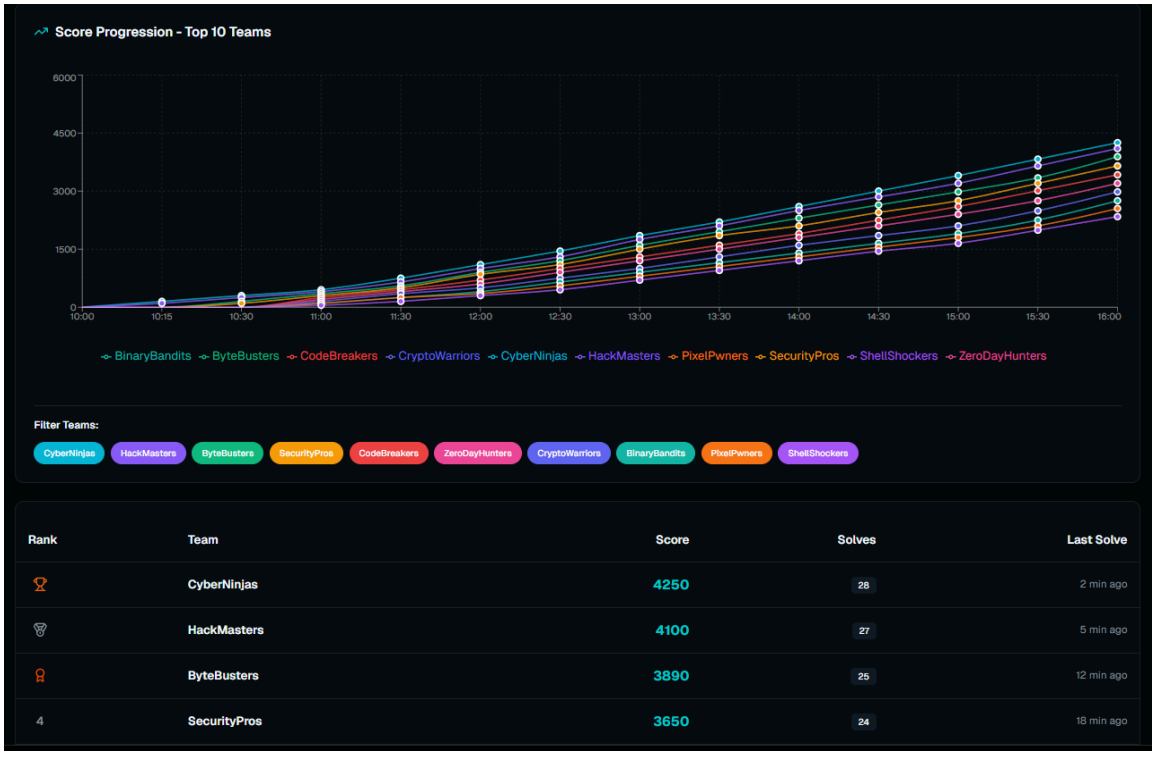


2.3 Leaderboard Page

The Leaderboard section provides a dynamic and comprehensive view of team performance throughout the competition, serving as the primary source for tracking progress and fostering a competitive environment. It is designed to present complex ranking data through an interactive graphical representation and a detailed summary table.

The core of the interface is the "Score Progression" graph, a multi-line chart that visualizes the score accumulation of the top ten teams over time. This visualization allows participants to analyze the pace and historical performance of the leading contenders. A key feature of this graph is its interactivity; a dedicated legend listing all charted teams allows users to selectively show or hide any team's progression line by selecting its name, enabling a less cluttered and more focused analysis of specific rivalries or performances. Furthermore, to enhance situational awareness for the participant, the progression line representing their own team is rendered with a distinct, bolder stroke, ensuring it is immediately identifiable amidst the other teams.

Alongside the graphical overview, a table presents a precise, numerical snapshot of the current standings. This table lists teams with their corresponding metrics: overall score, total number of challenges solved, and the time elapsed since their last successful submission. This combination of a customizable historical graph and a real-time table offers participants multiple, complementary perspectives on the competitive landscape of the event.

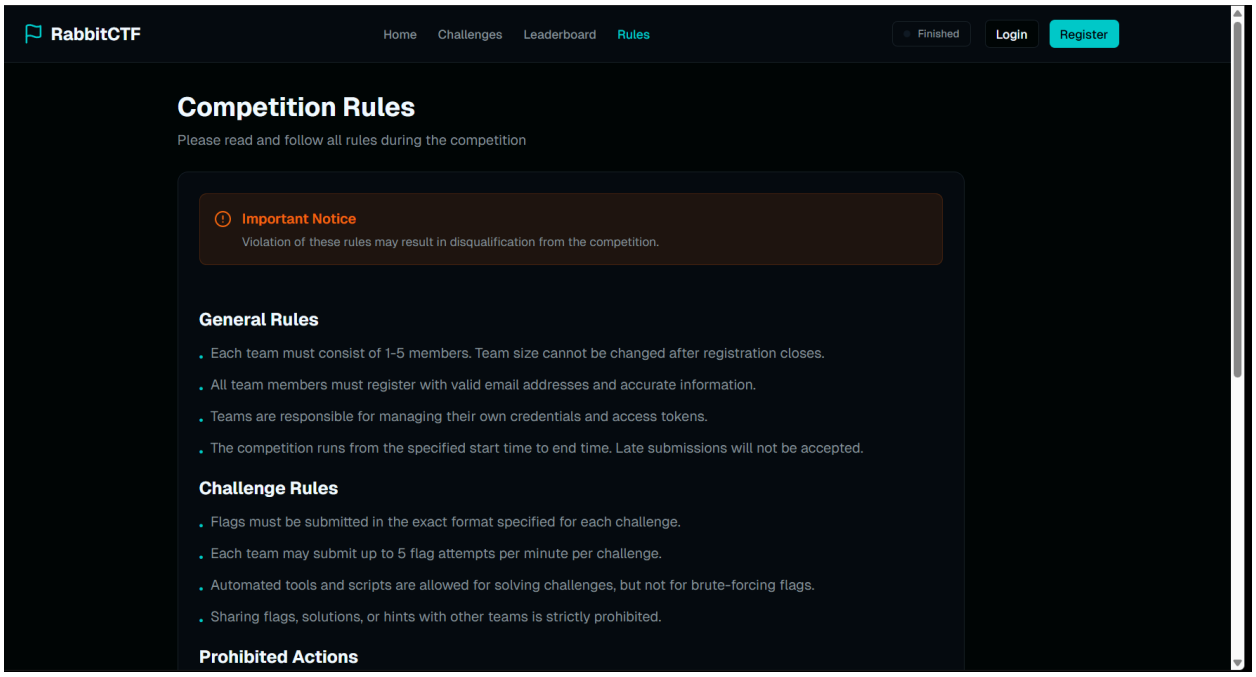


2.4 Rules Page

The Rules page serves as the official repository for the competition's guidelines and regulations, providing a centralized and static reference for all participants. Its design prioritizes clarity and readability, ensuring that the critical information governing the event is accessible and unambiguous.

The page presents a clean, structured document with a clear title, "Competition Rules," followed by a prominent notice emphasizing the importance of adherence and the consequences of violations. The content is organized into logical sections, such as "General Rules" and "Challenge Rules," utilizing bulleted lists to present each individual rule in a scannable and digestible format. This formatting enhances the usability of the document, allowing participants to quickly locate specific stipulations regarding team composition, registration, submission formats, and code of conduct.

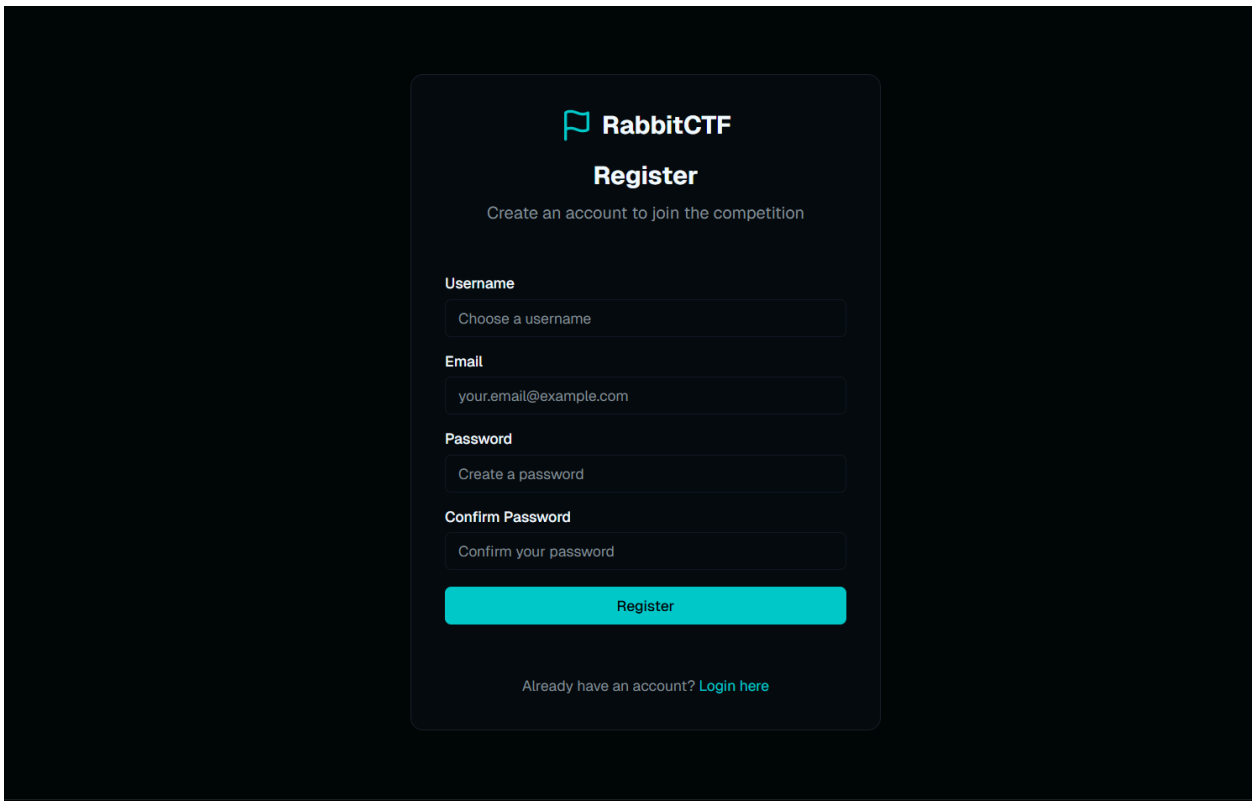
The content itself is dynamically rendered from a source managed exclusively through the administrative panel. This design ensures that all participants view a consistent, authoritative, and up-to-date version of the rules, which administrators can modify as needed using a Markdown editor. The interface is purely for consumption, with no editing capabilities available to competitors, thereby maintaining the integrity and official nature of the published regulations.



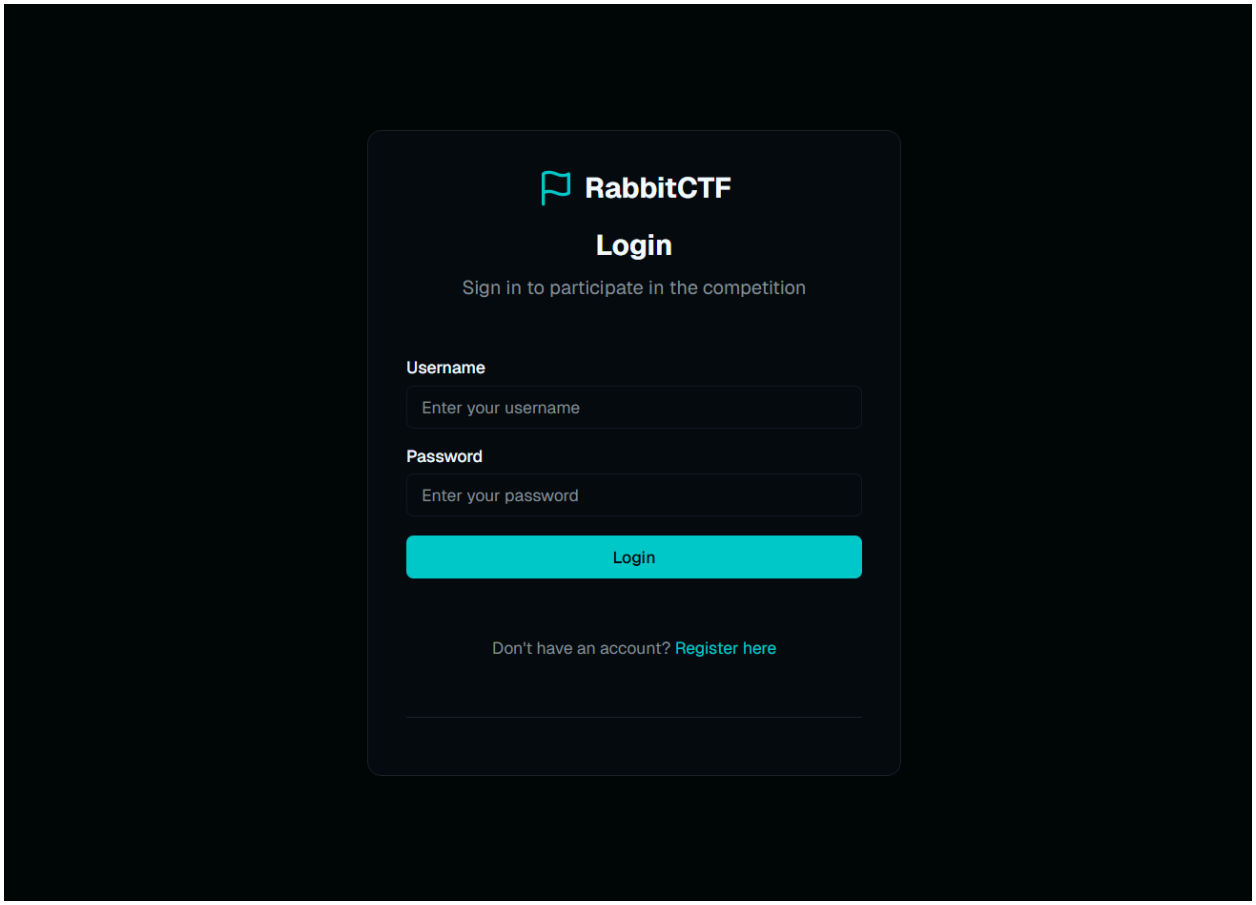
2.5 Login and Register Pages

The authentication section provides the essential gateway for user participation in the RabbitCTF platform, comprising two distinct yet complementary interfaces for user registration and login. Both pages maintain a consistent and minimalist design, focusing user attention on the core task of account management without unnecessary distraction.

The registration interface presents a form for creating a new account, requesting the fundamental information required for user identification and system access. The form fields include a username, a valid email address, and a password, which must be confirmed in a dedicated field to prevent input errors. A prominent "Register" button serves as the call to action for submitting the completed form. To facilitate navigation between the two authentication states, a link is provided at the bottom of the page, allowing prospective users who already possess an account to transition seamlessly to the login page.



The login interface mirrors the streamlined aesthetic of the registration page, featuring a simplified form that collects the user's credentials: username and password. A "Login" button initiates the authentication process. Consistent with the registration page, a navigational link is available for users who have not yet created an account, enabling them to easily access the registration form. This paired design ensures a fluid and intuitive user experience for both new and returning participants, effectively managing their entry into the competitive environment.



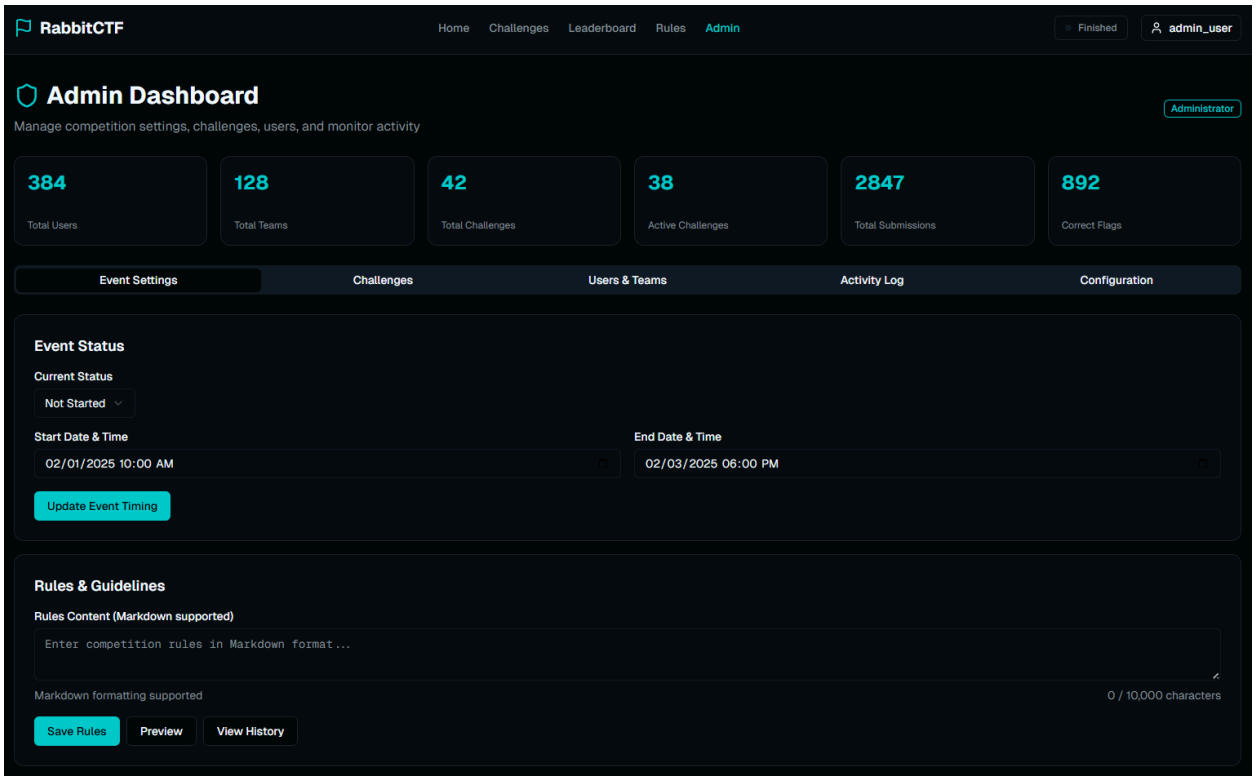
2.6 Admin Page

The Administrator Panel is a centralized and comprehensive management interface for the RabbitCTF platform. Its design is structured around a primary horizontal navigation bar that provides logical access to all administrative functions. The main content area dynamically

updates to present the specific tools, data, and configuration options relevant to the selected top-level category, enabling efficient oversight and control of the entire competition.

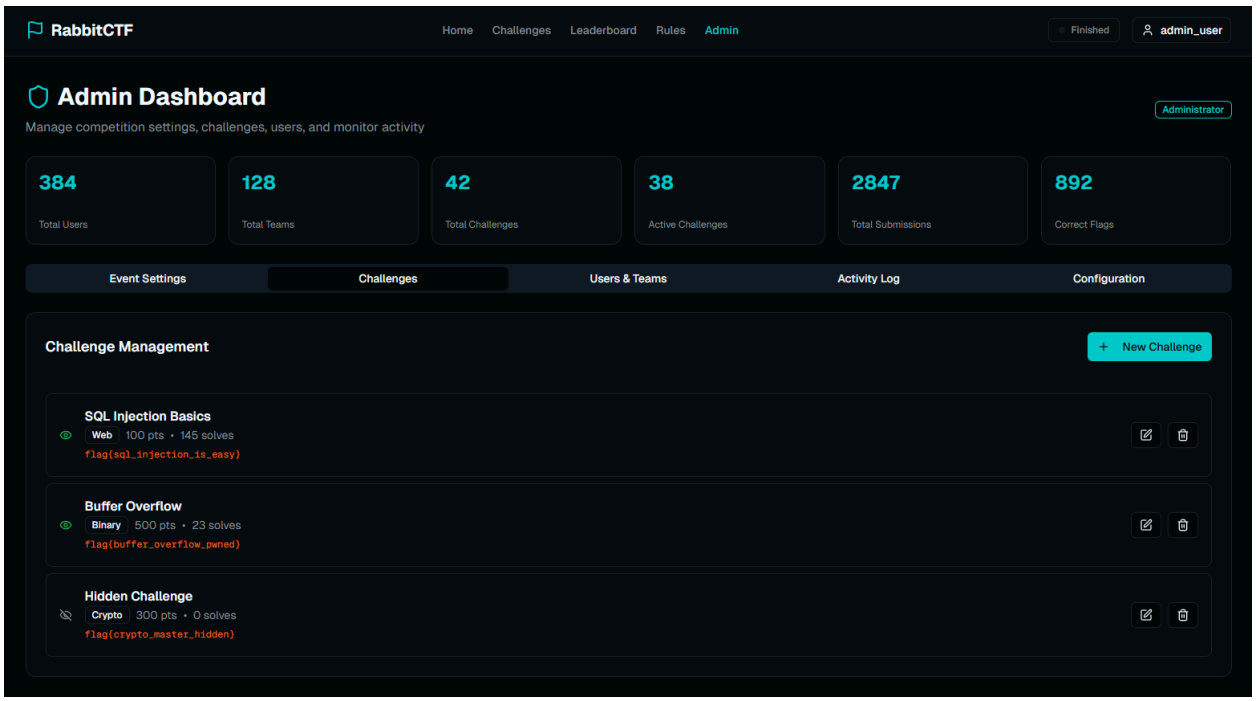
2.6.1 Event Configuration

This section serves as the primary dashboard and central hub for controlling the competition's core parameters. Upon selection, it presents an overview through a set of key metric cards, displaying real-time data such as Total Users, Registered Teams, Total and Active Challenges, Total Submissions, and Correct Flags. A central widget, the "Event Status" panel, provides direct control over the competition's lifecycle, allowing administrators to define the precise start and end timestamps and manually transition the event between "Not Started," "Active," and "Finished" states. Adjacent to this, the "Rules & Guidelines" module features a dedicated Markdown text editor with a live character counter and formatting toolbar. This interface supports the creation, editing, and version-controlled management of the competition's official rules, including options to save drafts, preview the rendered content, and review the edit history.



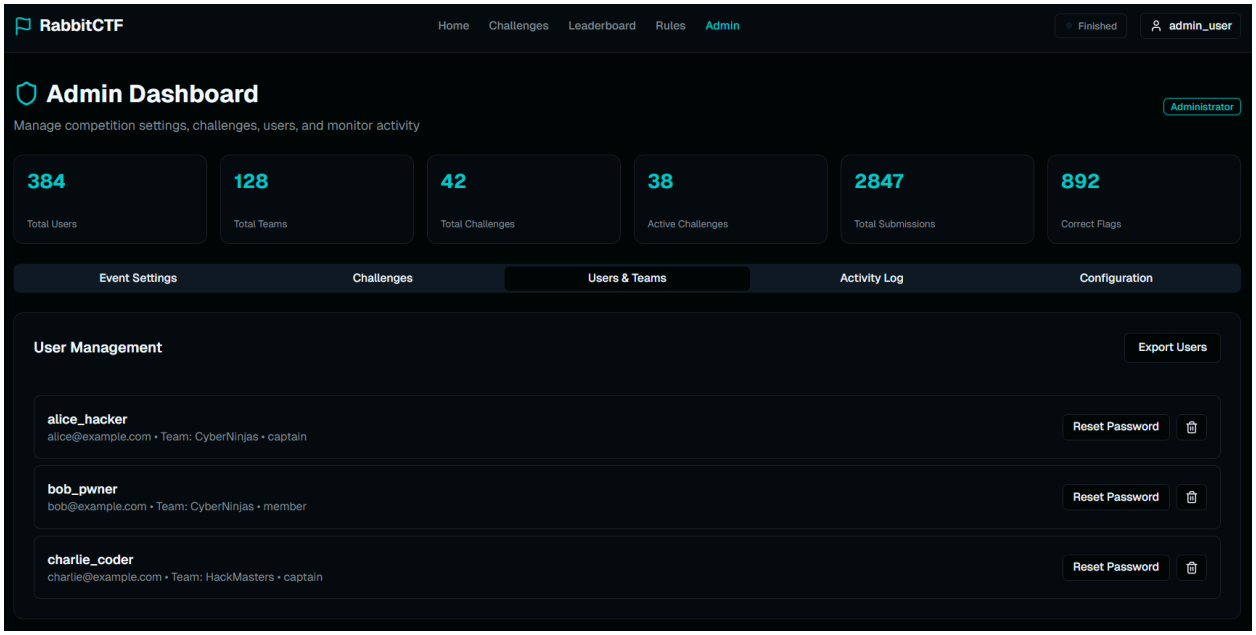
2.6.2 Challenges

This module is dedicated to the end-to-end lifecycle management of all competition challenges. It presents a sortable and filterable table listing all existing challenges, with columns displaying key attributes like title, category, difficulty, point value, solve count, and visibility status. Action buttons within this list enable immediate editing or deletion of individual challenges. A "New Challenge" button provides access to a comprehensive creation form. This form allows administrators to define all challenge properties, including its title, detailed description, category assignment, difficulty level, and tags. Furthermore, it includes fields for setting the flag value (with case sensitivity options), configuring the scoring mode (static or dynamic with configurable decay parameters), attaching resource files, and inputting connection information for remote services. A critical function within this form is the ability to set the challenge's initial visibility to either "draft" or "published."



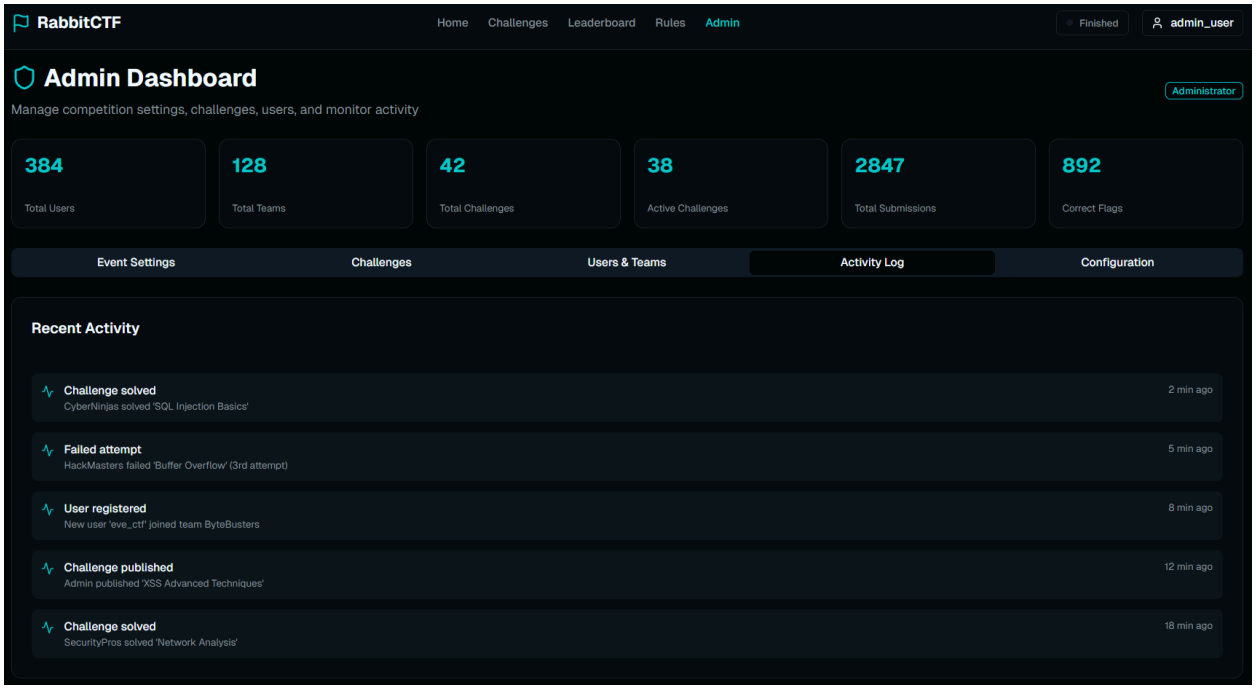
2.6.3 Users & Teams

This section facilitates detailed administration of all participants and their groupings. The "User Management" pane displays a searchable and sortable list of all registered users, showing their username, email address, and associated team. For each user, contextual actions are available, most notably the ability to reset their password, which is essential for user support. The "Team Management" pane provides a parallel interface for all registered teams. It lists team names, their members (highlighting the team captain), and their accumulated scores. This area provides administrative capabilities to view team details, manage membership, or, if necessary, dissolve a team, which would automatically unassign all its members.



2.6.4 Activity Log

This module functions as a real-time audit trail and system monitoring console. It presents a continuous, timestamped feed of significant events within the platform. Log entries are categorized and detail actions such as user registrations, successful and failed flag submission attempts (without logging the flag value itself), challenge solves, administrative publications of new challenges, and changes to team rosters. This chronological log is indispensable for tracking event progress, diagnosing participant issues, identifying potential rule violations, and maintaining a security audit trail.

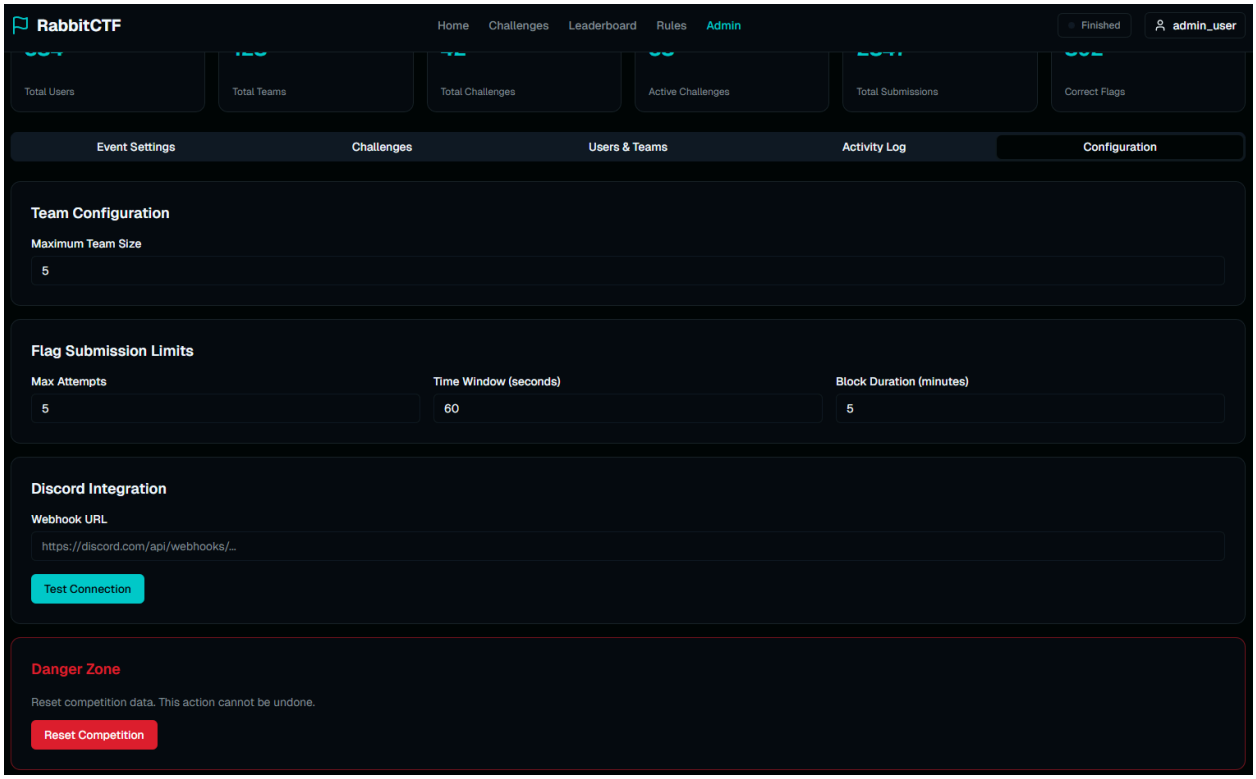


2.6.5 Settings

This category aggregates the configuration of global system policies and external integrations. It is organized into several specialized sub-panes:

- **Team Configuration:** Contains a single setting to define the maximum number of members allowed per team, which is enforced during team creation and when members attempt to join.
- **Submission Limits:** Provides a set of inputs to define a global rate-limiting policy for flag submissions. This includes the maximum number of attempts allowed per user, the time window for those attempts, and the duration of a temporary block imposed if the limit is exceeded.
- **Discord Integration:** Features a field for inputting a Discord webhook URL to enable external notifications for events like new challenge publications or "first blood" achievements. A "Test Connection" button allows for validation of the webhook's functionality.

- **Danger Zone:** A visually distinct section reserved for high-impact, irreversible actions. The most critical function here is "Reset Competition," which triggers a complete or partial wipe of competition data (e.g., submissions, scores) according to a predefined policy. This action is typically guarded by multiple confirmation prompts to prevent accidental use.



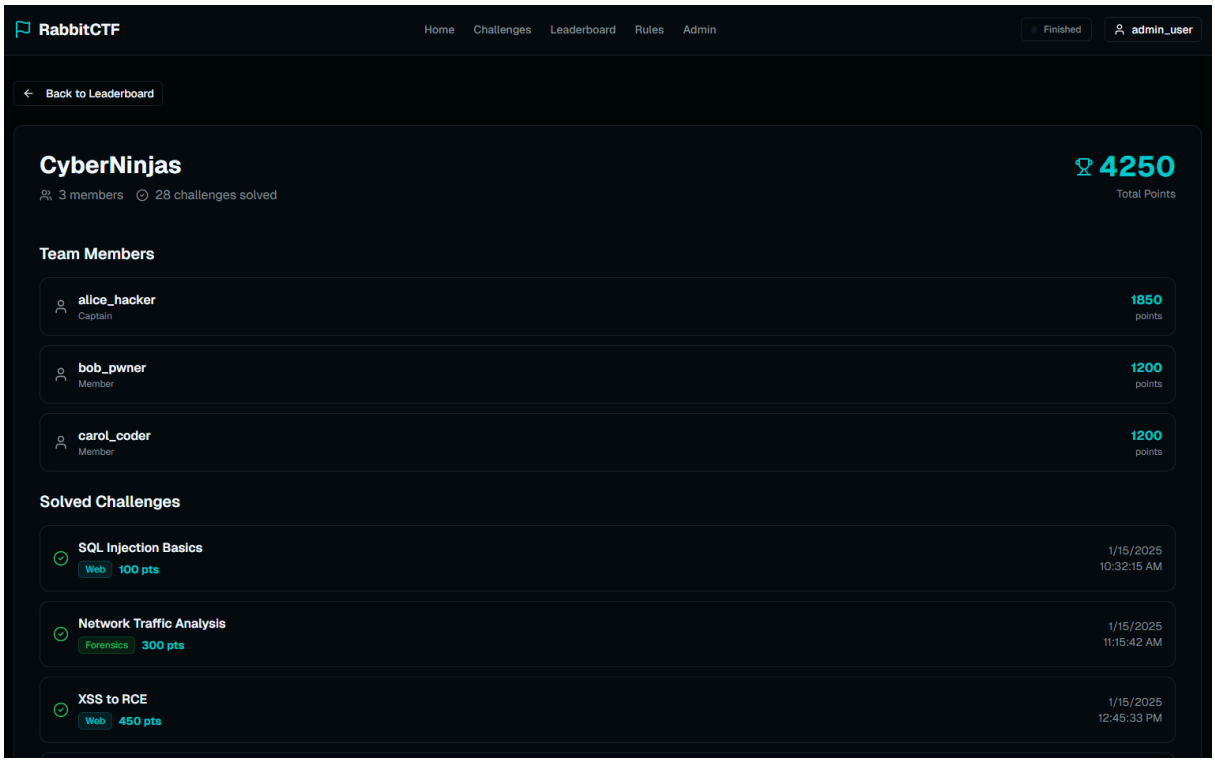
This detailed and logically segmented structure ensures that administrators can efficiently and securely manage every facet of the CTF event, from broad-stroke configuration to granular user and challenge oversight.

2.7 Team Page

The Team Profile page provides a detailed, public-facing overview of a specific team's composition and performance within the competition. Accessible directly from the Leaderboard, this page offers a consolidated view of a team's identity, membership, and accomplishments, allowing for the assessment of rival teams and reinforcing team identity for its own members.

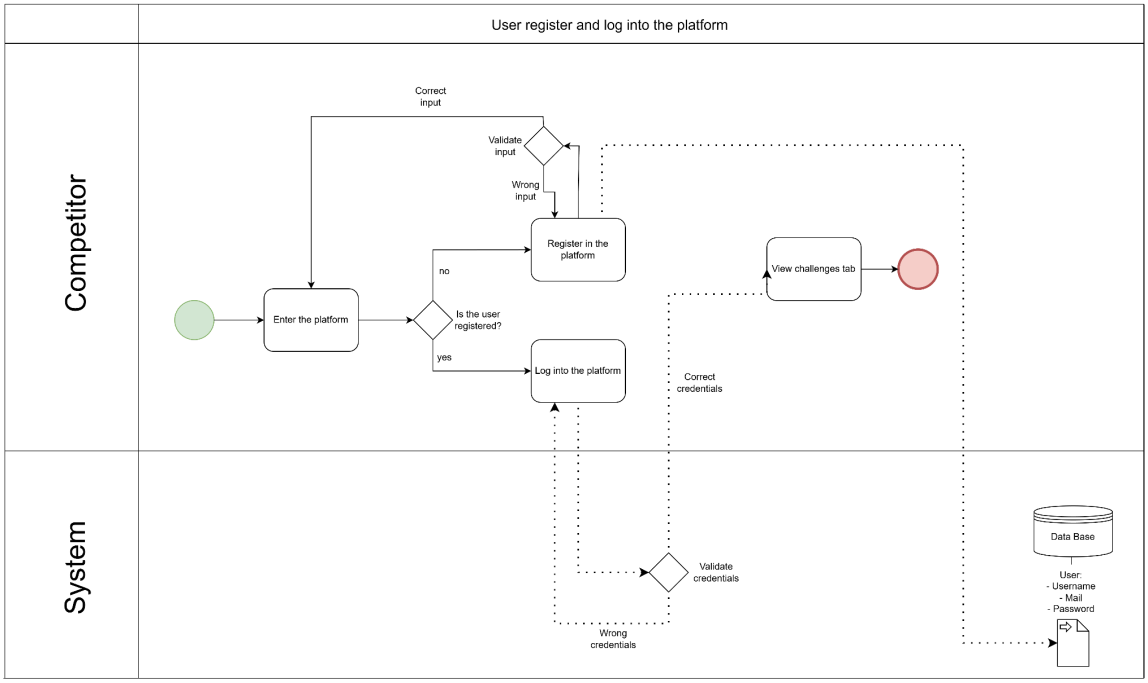
The page is organized into distinct, horizontal rows that segment the information logically. The top section is dedicated to the team's identity and structure, prominently displaying the team's name followed by a "Team Members" list. This list details each member's username and clearly indicates their role, distinguishing between the team captain and regular members to provide immediate insight into the team's internal organization.

Beneath this, a larger "Solved Challenges" section presents a chronological record of the team's successes. Each solved challenge is listed in its own row, displaying the challenge title, its category, the points awarded, and the precise date and timestamp of the solution. This creates a visual timeline of the team's progress and achievements throughout the event. The section is headed by a prominent display of the team's total accumulated score, serving as a definitive summary of their overall performance. This row-based layout effectively presents a comprehensive and transparent profile of a team's presence and prowess within the competition.



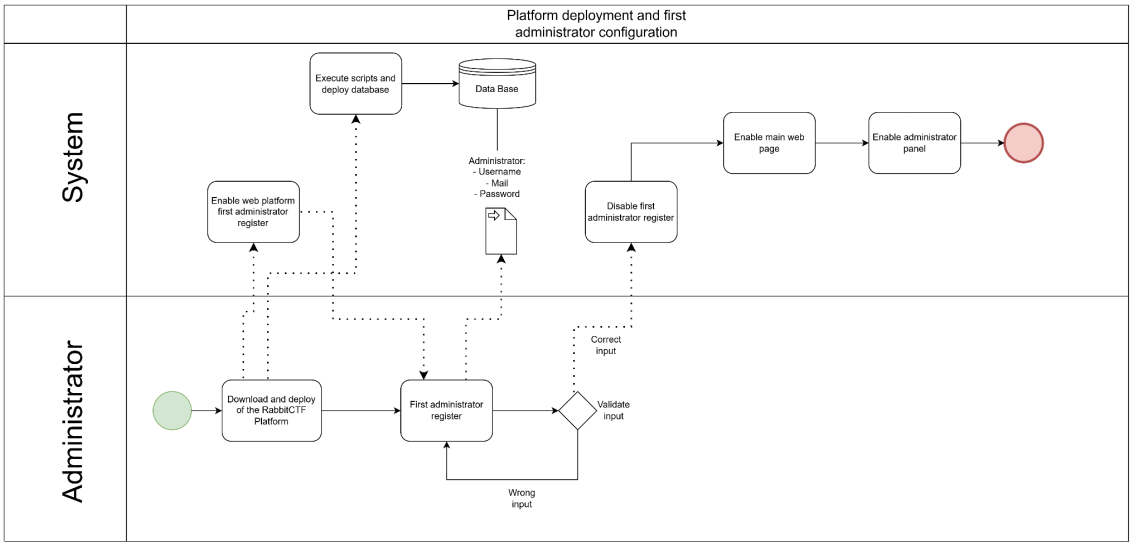
3 Business Model Processes

3.1 User register and Log Into the Platform



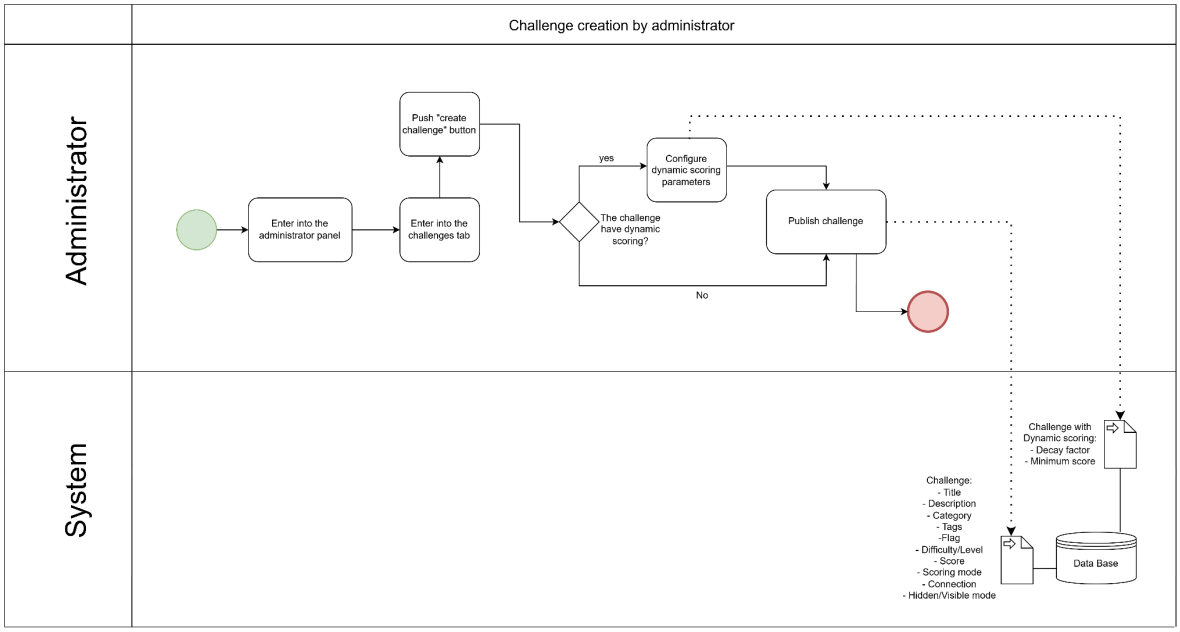
The user registration and login process allows competitors to access the CTF platform securely. A new user first enters the platform and submits registration information, which the system validates before storing it in the database. Registered users can then log in by providing their credentials, which the system checks against stored data. Upon successful authentication, the competitor gains access to the challenges tab, where they can view available tasks. This process provides controlled, authenticated access to the platform while ensuring user information is handled securely.

3.2 Platform Deployment and First Administrator Configuration



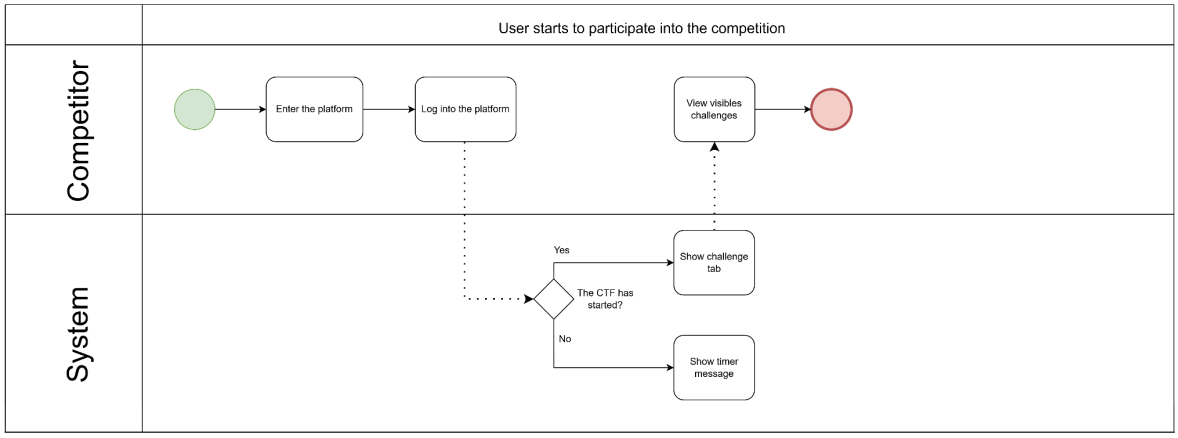
The platform deployment and first administrator configuration process establishes the initial operational state of the CTF platform. The administrator begins by downloading and deploying the platform software, after which the system executes setup scripts and creates the database. The admin then registers the first administrator account, with the system validating input and storing these credentials. Once this setup is complete, the system enables the main webpage and administrator panel and disables additional first-time admin registrations to protect the platform. This process ensures a secure foundation and proper administrative control before competition management begins.

3.3 Challenge Creation by Administrator



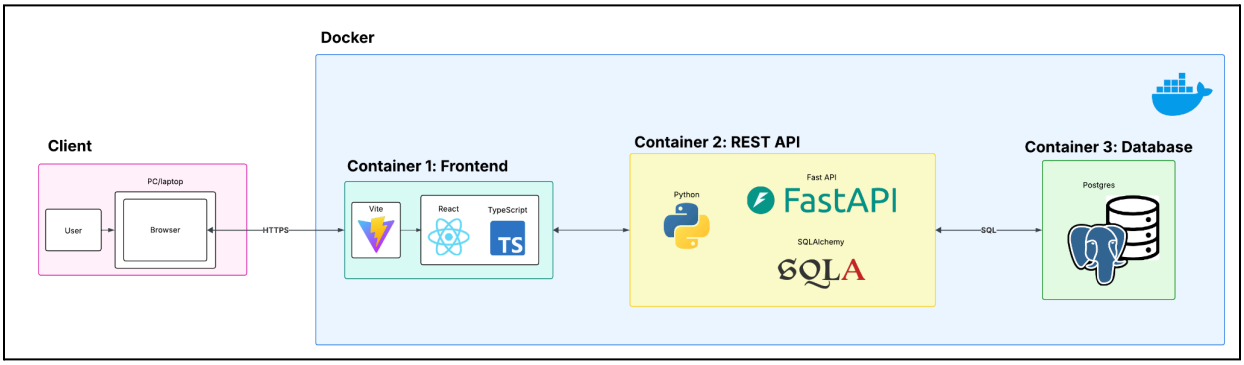
The challenge creation process enables administrators to design and publish tasks that competitors will solve during the CTF. After accessing the admin panel and entering the challenges section, the administrator selects the option to create a new challenge and fills in key details such as title, description, category, difficulty, and flag. If dynamic scoring is desired, the administrator further configures parameters like decay rate and minimum score. The system then stores the completed challenge in the database and publishes it to the platform. This process supplies the competition with structured, properly configured challenges for participants.

3.4 User Starts to Participate into the Competition



The user participation process governs how competitors interact with challenges once the event is active. After entering the platform and logging in, the system checks whether the competition has officially started. If not, the user is shown a countdown or waiting message; if it has begun, the system displays all visible challenges. Competitors can then browse and attempt to solve the tasks available to them. This process ensures fair access to challenges according to event timing and supports smooth engagement during the CTF.

4 Architecture Diagram



The system is structured according to a three-tier, container-based architecture, comprising a frontend client, a REST API backend, and a dedicated database. This design promotes a clear separation of concerns, modularity, and facilitates secure, efficient communication between components, all orchestrated within a Docker environment.

At the client tier, users interact with the application via a web browser. The frontend is implemented as a single-page application using React with TypeScript, ensuring a robust and maintainable codebase for building a dynamic user interface. This container is responsible for rendering all visual components, managing user interactions, and issuing HTTP requests to the backend API. It serves as the sole presentation layer, with no business logic or direct data persistence.

The application logic is centralized in the REST API tier, which is built using Python and the FastAPI framework. This container exposes a structured set of endpoints for all platform operations, including user authentication, challenge management, and flag validation. Internally, it utilizes SQLAlchemy as an Object-Relational Mapper (ORM) to abstract and handle all data interactions. This service processes business rules, validates inputs, and acts as the mandatory gateway for all data access, ensuring security and consistency.

The data persistence tier is managed by a PostgreSQL database container. It is solely responsible for the secure storage and retrieval of all application data, such as user profiles, challenge details, team information, and submission records. Communication between the REST API and the database is conducted through the SQLAlchemy library, which translates object-oriented operations into efficient SQL queries. This tier is isolated within the Docker

network, with no direct exposure to the public internet, safeguarding the integrity and confidentiality of the data.

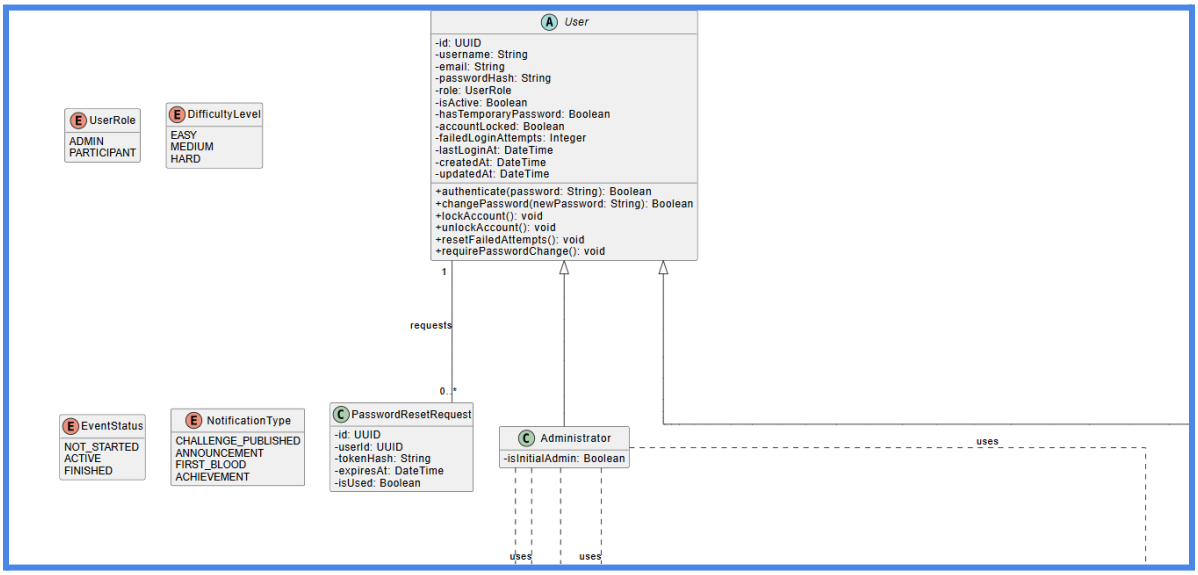
Docker Compose orchestrates the entire system, defining the services, their dependencies, and the internal network that enables secure communication between the frontend, backend, and database containers. This containerized approach guarantees environmental consistency, simplifies deployment and scaling procedures, and provides strong isolation between the system's functional layers. The data flow is unidirectional: user actions in the frontend trigger API calls to the backend, which in turn performs the necessary operations on the database, with the results being returned along the same path to the user.

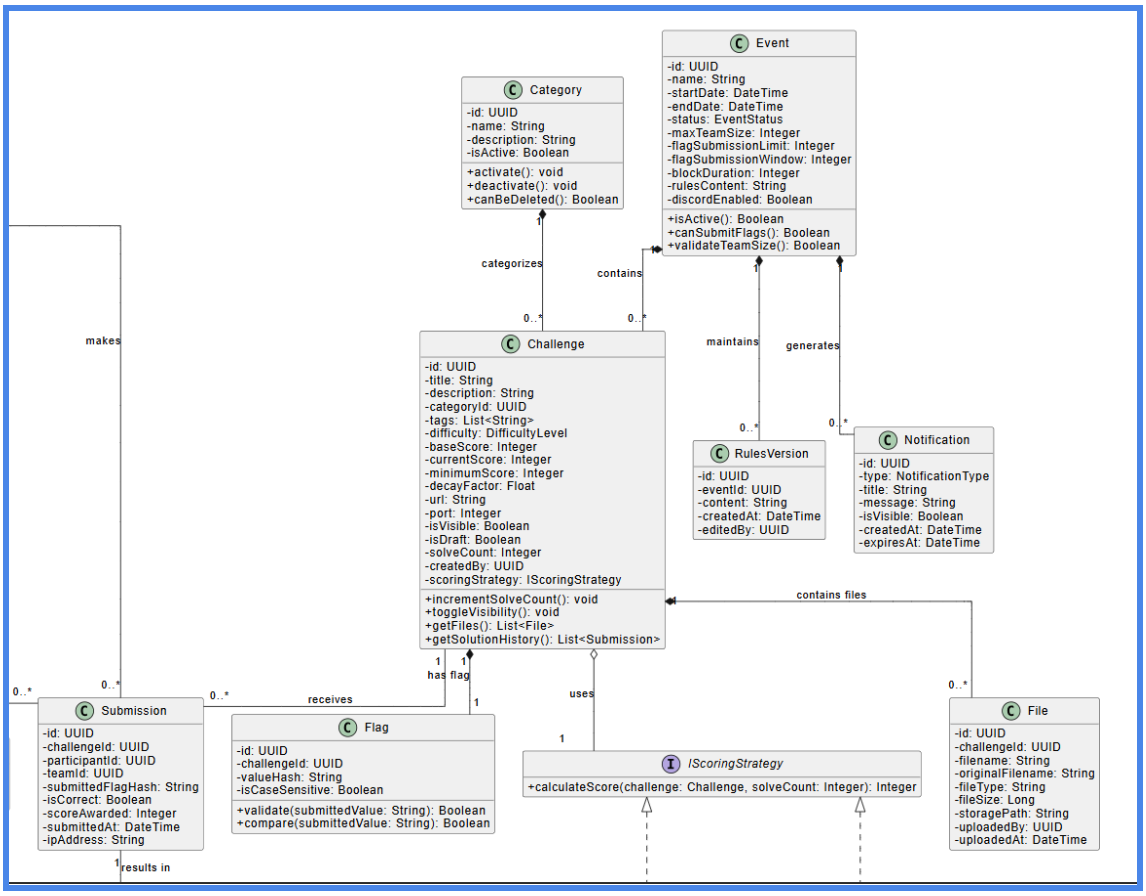
5 Class Diagram

Due to the big size of the class diagram it is contained in a .svg file where it looks clear. In the next share link is the Class diagram:

class_diagram.svg

The next images show some parts of the class diagram:





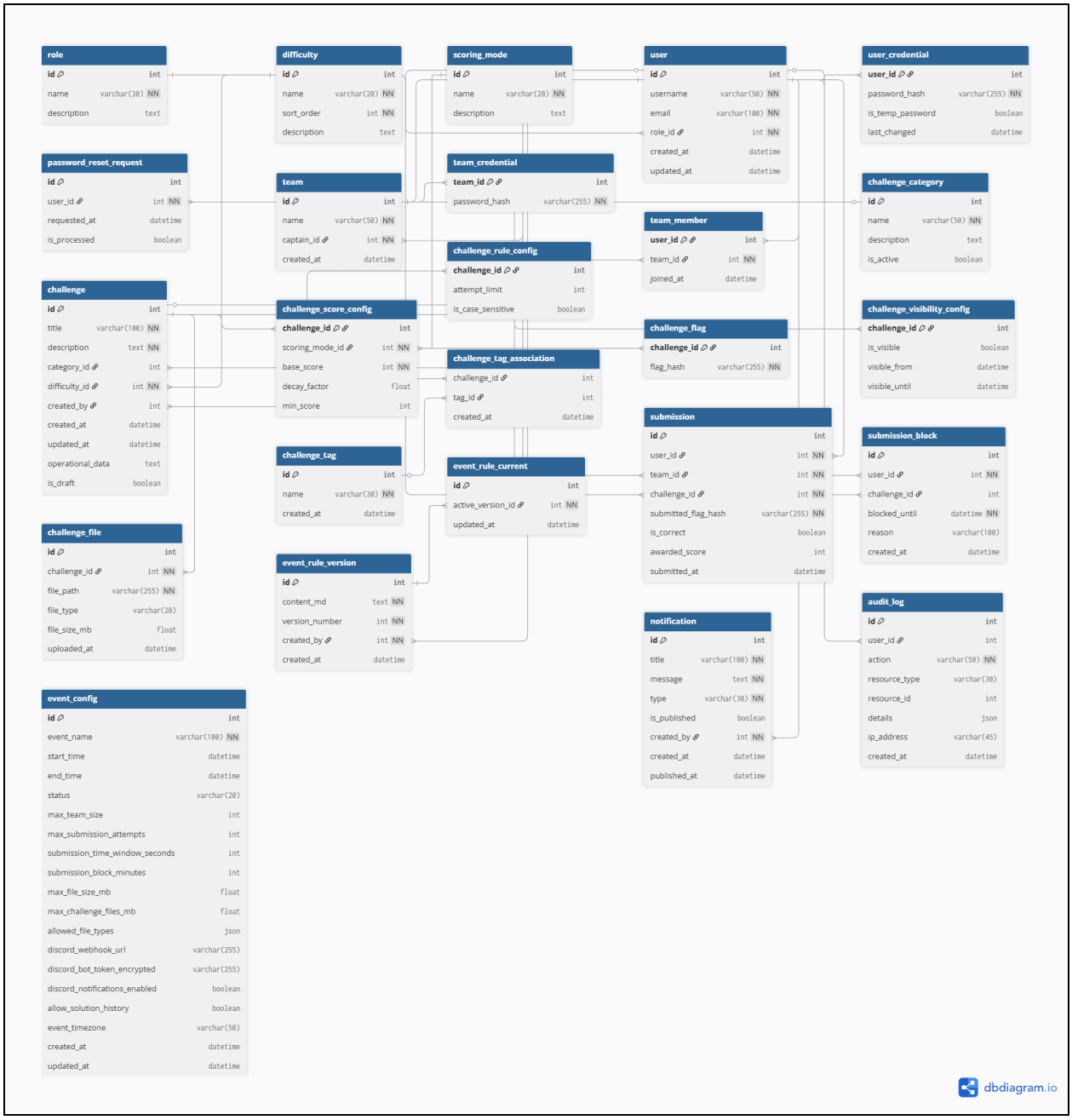
For the design and modeling of the CTF platform, a UML class diagram was developed based on the CRC cards defined earlier in the project. Each class in the diagram includes its attributes, methods, and visibility, providing a clear view of the system's structure and the responsibilities of each component. The diagram reflects the main entities of the platform, such as users, teams, challenges, submissions, notifications, and events, which together define the core functionality of a CTF competition system.

In addition, different types of UML relationships were included to represent how the system components interact. The diagram uses generalization (inheritance) to show how the Administrator and Participant classes extend the abstract User class. It also includes associations with specific multiplicities, such as the relationship where a Team can have one or more Participants, or a Category can contain multiple Challenges. In some cases, aggregation was used to indicate "whole–part" relationships where the parts can exist independently, such as a Challenge having multiple Files.

The diagram also reflects important one-to-many associations between core components, such as Participant to Submission and Challenge to Flag, emphasizing how each submission relates to a single challenge and team. These relationships help define the flow of data within the system and support features like scoring, leaderboard calculation, and event management. In conclusion this Class diagram establishes a good for the implementation of the system in the next stage.

6 Relational Database Model

The RabbitCTF database schema represents a comprehensive data architecture designed to support the operational requirements of a competitive cybersecurity platform. This relational model facilitates user management, team dynamics, challenge administration, real-time scoring mechanisms, and event coordination within a unified framework. The schema employs a normalized structure while incorporating strategic denormalization to optimize performance-critical operations such as leaderboard calculations and submission processing. The design philosophy emphasizes data integrity, security through separation of concerns, and scalability to accommodate fluctuating participant volumes during active competition periods.



6.1 Core Reference Data

The foundation of the schema rests upon systematically organized reference data that ensures consistency across the platform. The role table establishes a hierarchical privilege system that governs user capabilities and access permissions throughout the application. Difficulty classifications provide a standardized spectrum of challenge complexity, incorporating both

descriptive labels and numerical ordering to support flexible presentation logic. The scoring_mode table defines fundamental valuation methodologies, distinguishing between static point assignments that remain constant throughout the competition and dynamic scoring models that algorithmically adjust based on solution frequency and timing. These reference tables serve as authoritative sources for enumerable values, maintaining referential integrity while allowing administrative customization of the competitive framework.

6.2 User Management and Authentication

User identity and authentication are implemented through a segregated architecture that enhances security while maintaining operational flexibility. The user table functions as the central identity repository, storing essential profile information and establishing relationships to privilege levels. Authentication credentials are maintained in a distinct user_credential table that employs cryptographic hashing for password storage, with additional fields tracking temporary password status and credential modification timestamps. This separation limits exposure of sensitive authentication data during routine user operations. The password_reset_request table provides an administrative workflow for credential recovery, maintaining request audit trails and processing status to ensure accountable resolution of access issues.

6.3 Team Organization Structure

Team management implements a sophisticated relationship model that balances organizational flexibility with competitive integrity. The team table serves as the primary organizational entity, incorporating a direct reference to a team captain through a foreign key constraint that enforces single-captain ownership while preventing users from leading multiple teams simultaneously. Team access control is managed through separate credential storage in team_credential, allowing distinct authentication mechanisms for team administration. The team_member association table governs participation by enforcing exclusive membership through primary key constraints, ensuring competitors cannot join multiple teams during a single event. This membership model supports dynamic team formation while maintaining fair competition through exclusivity guarantees.

6.4 Challenge Management System

Challenges constitute the core competitive elements within the platform, with a multi-table architecture that supports complex configuration requirements. The challenge table maintains descriptive metadata including titles, detailed descriptions, and operational data such as network endpoints or connection parameters. Relationships to challenge_category and difficulty tables enable systematic organization and discovery. Challenge behavior is governed through specialized configuration tables: challenge_score_config manages scoring parameters including base values, decay algorithms, and minimum point thresholds; challenge_rule_config defines submission limitations and validation rules such as case sensitivity; challenge_visibility_config controls temporal availability through scheduled visibility windows. Security-sensitive flag data is isolated within challenge_flag using cryptographic hashing to prevent exposure through database introspection or application vulnerabilities.

6.5 Supplementary Challenge Components

Beyond core challenge configuration, the schema provides extensive support for supplementary materials and advanced categorization. The challenge_file table manages associated file attachments with comprehensive metadata including storage paths, file types, and size information, facilitating download management and storage quota enforcement. A flexible tagging mechanism through challenge_tag and challenge_tag_association tables enables multi-dimensional classification beyond primary category organization, supporting advanced filtering and discovery interfaces. These components enhance the educational value of challenges by providing relevant resources while maintaining organizational structure through systematic metadata management.

6.6 Competition Participation Tracking

The submission system provides a comprehensive framework for recording solution attempts while maintaining competitive integrity and security. The submission table serves as the

central audit trail for all flag validation attempts, capturing user and team associations, challenge references, and temporal metadata. Security is enhanced through storage of hashed submission values rather than plaintext flags, preventing exposure in logs or database exports. Awarded scores are denormalized within submission records to preserve historical accuracy despite potential changes in challenge valuation parameters. The submission_block table implements rate limiting enforcement by tracking temporary restrictions imposed upon users who exceed configured attempt thresholds, providing protection against brute-force attacks while maintaining service availability.

6.7 Event Configuration and Management

Runtime competition parameters are centralized within the event_config table, which serves as the primary administrative interface for operational customization. This comprehensive configuration repository manages temporal boundaries defining competition duration, team size restrictions, and submission rate limiting parameters. File management policies including size constraints and permitted file types are enforced through stored configuration values. External integration settings for platforms such as Discord are maintained alongside core competition parameters. The event_rule_version and event_rule_current tables implement a versioned content management system for competition rules and guidelines, preserving full revision history while maintaining a clear reference to currently active documentation.

6.8 Communication and Audit Systems

The notification table provides a broadcast communication channel for competition announcements, administrative updates, and system alerts through a simplified forum-style model. This approach ensures consistent dissemination of information to all participants without complex targeting or read receipt requirements. Comprehensive activity monitoring is implemented through the audit_log table, which captures user actions, resource modifications, contextual metadata, and originating addresses for security analysis, operational troubleshooting, and compliance reporting. This audit infrastructure supports retrospective analysis of competition dynamics and security incident investigation.

6.9 Structural Integrity and Performance

The schema incorporates extensive indexing strategies specifically optimized for common access patterns, particularly those supporting real-time leaderboard calculations, challenge discovery interfaces, and submission history retrieval. Relationship integrity is enforced through foreign key constraints while business rules are implemented through unique constraints and application-level validation. The balanced approach to normalization ensures data consistency without sacrificing performance for high-frequency operations such as flag submission and score updates. This architectural consideration provides a robust foundation capable of maintaining responsive performance during peak competition activity while ensuring accurate results for scoring and ranking calculations.

6.10 Diagram with DBML

Given the complexity of the database schema, capturing it in a single image would make it difficult to review and understand each individual component and their relationships. Therefore, the DBML code is provided below to facilitate its reconstruction at https://dbdiagram.io/d/. This tool allows for an interactive visualization and a more granular examination of the tables, columns, and references.

```
SQL
//////////////////////////////////////////////////////
// RABBITCTF DATABASE SCHEMA
//////////////////////////////////////////////////////

//////////////////////////////////////////////////////
// REFERENCE DATA (ENUMS AS TABLES)
//////////////////////////////////////////////////////

Table role {
```

```
  id              int [pk, increment]
  name            varchar(30) [unique, not null]
  description     text

  indexes {
    name [unique]
  }
}

Table difficulty {
  id              int [pk, increment]
  name            varchar(20) [unique, not null] // EASY, MEDIUM, HARD
  sort_order      int [not null]
  description     text

  indexes {
    name [unique]
    sort_order
  }
}

Table scoring_mode {
  id              int [pk, increment]
  name            varchar(20) [unique, not null] // STATIC, DYNAMIC
  description     text

  indexes {
    name [unique]
  }
}

////////////////////////////////////////////////////
// USERS & AUTHENTICATION
////////////////////////////////////////////////////
```

```
Table user {
  id              int [pk, increment]
  username        varchar(50) [unique, not null]
  email           varchar(100) [unique, not null]
  role_id         int [not null, ref: > role.id]
  created_at      datetime [default: `now()`]
  updated_at      datetime

  indexes {
    username [unique]
    email [unique]
    role_id
    created_at
  }
}

Table user_credential {
  user_id         int [pk, ref: > user.id]
  password_hash   varchar(255) [not null]
  is_temp_password boolean [default: false]
  last_changed    datetime [default: `now()`]
}

Table password_reset_request {
  id              int [pk, increment]
  user_id         int [not null, ref: > user.id]
  requested_at    datetime [default: `now()`]
  is_processed    boolean [default: false]

  indexes {
    user_id
    requested_at
    is_processed
```

```
  }
}

//////////////////////////////////////////////////////
// TEAMS & MEMBERSHIP
//////////////////////////////////////////////////////

Table team {
  id              int [pk, increment]
  name            varchar(50) [unique, not null]
  captain_id      int [not null, ref: > user.id, unique]
  created_at      datetime [default: `now()`]

  indexes {
    name [unique]
    captain_id [unique]
    created_at
  }

  Note: "Each team has one captain, and a user can be captain of only one team"
}

Table team_credential {
  team_id         int [pk, ref: > team.id]
  password_hash   varchar(255) [not null]
}

Table team_member {
  user_id         int [pk, ref: > user.id]
  team_id         int [not null, ref: > team.id]
  joined_at       datetime [default: `now()`]

  indexes {
    team_id
```

```
    (user_id, team_id) [unique]
  }

  Note: "User can only be on one team - user_id is primary key"
}

////////////////////////////////////////////////////
// CHALLENGES & CATEGORIES
////////////////////////////////////////////////////

Table challenge_category {
  id              int [pk, increment]
  name            varchar(50) [unique, not null]
  description     text
  is_active       boolean [default: true]

  indexes {
    name [unique]
    is_active
  }
}

Table challenge {
  id              int [pk, increment]
  title           varchar(100) [not null]
  description     text [not null]
  category_id     int [ref: > challenge_category.id]
  difficulty_id   int [not null, ref: > difficulty.id]
  created_by      int [ref: > user.id]
  created_at      datetime [default: `now()`]
  updated_at      datetime
  operational_data text // For URLs, ports, connection info (FR-3.23)
  is_draft        boolean [default: true] // For visibility control (FR-2.2)
```

```
  indexes {
    category_id
    difficulty_id
    created_by
    (category_id, is_draft, difficulty_id)
    created_at
  }
}

Table challenge_score_config {
  challenge_id    int [pk, ref: > challenge.id]
  scoring_mode_id int [not null, ref: > scoring_mode.id]
  base_score      int [not null]
  decay_factor    float
  min_score       int

  Note: "Current score calculated from base_score, solves, and decay"
}

Table challenge_rule_config {
  challenge_id    int [pk, ref: > challenge.id]
  attempt_limit   int [default: 5]
  is_case_sensitive boolean [default: true]
}

Table challenge_flag {
  challenge_id    int [pk, ref: > challenge.id]
  flag_hash       varchar(255) [not null]

  Note: "Separated for security - restrict access to this table"
}

Table challenge_visibility_config {
  challenge_id    int [pk, ref: > challenge.id]
```

```
  is_visible      boolean [default: false]
  visible_from    datetime
  visible_until   datetime

  indexes {
    is_visible
    (is_visible, visible_from, visible_until)
  }
}

Table challenge_file {
  id              int [pk, increment]
  challenge_id    int [ref: > challenge.id, not null]
  file_path       varchar(255) [not null]
  file_type       varchar(20)
  file_size_mb    float
  uploaded_at     datetime [default: `now()`]

  indexes {
    challenge_id
    uploaded_at
  }
}

////////////////////////////////////////////////////
// TAGS & FILTERING
////////////////////////////////////////////////////

Table challenge_tag {
  id              int [pk, increment]
  name            varchar(30) [unique, not null]
  created_at      datetime [default: `now()`]

  indexes {
```

```
    name [unique]
  }
}

Table challenge_tag_association {
  challenge_id    int [ref: > challenge.id]
  tag_id          int [ref: > challenge_tag.id]
  created_at      datetime [default: `now()`]

  indexes {
    tag_id
    challenge_id
    (challenge_id, tag_id) [unique]
  }
}

////////////////////////////////////////////////////
// PARTICIPATION & FLAG SUBMISSIONS
////////////////////////////////////////////////////

Table submission {
  id              int [pk, increment]
  user_id         int [ref: > user.id, not null]
  team_id         int [ref: > team.id, not null]
  challenge_id    int [ref: > challenge.id, not null]
  submitted_flag_hash varchar(255) [not null] // For audit trail (FR-3.6)
  is_correct      boolean [default: false]
  awarded_score   int // Store the actual score awarded
  submitted_at    datetime [default: `now()`]

  indexes {
    user_id
    team_id
    challenge_id
```

```
    (team_id, is_correct)
    (user_id, challenge_id, submitted_at)
    (challenge_id, is_correct)
    submitted_at
  }

  Note: "Team denormalized for performance. Leaderboard calculated from correct
submissions."
}

Table submission_block {
  id              int [pk, increment]
  user_id         int [ref: > user.id, not null]
  challenge_id    int [ref: > challenge.id]
  blocked_until   datetime [not null]
  reason          varchar(100)
  created_at      datetime [default: `now()`]

  indexes {
    user_id
    challenge_id
    blocked_until
  }
}

////////////////////////////////////////////////////
// EVENT CONFIGURATION (Runtime configurable)
////////////////////////////////////////////////////

Table event_config {
  id              int [pk, increment]
  event_name      varchar(100) [not null]
  start_time      datetime
  end_time        datetime
```

```
  status           varchar(20) [default: 'not_started'] // not_started, active,
finished
  max_team_size   int [default: 4]

  // Submission rate limiting (FR-3.27-FR-3.32, FR-6.19-FR-6.21)
  max_submission_attempts int [default: 5]
  submission_time_window_seconds int [default: 60]
  submission_block_minutes int [default: 5]

  // File configuration (FR-2.9, FR-2.10)
  max_file_size_mb float [default: 100]
  max_challenge_files_mb float [default: 500]
   allowed_file_types json [default: '["zip", "tar.gz", "txt", "pdf", "pcap",
"png", "jpg"]']

  // Discord integration (FR-5.3, NFR-6.1-NFR-6.4)
  discord_webhook_url varchar(255)
  discord_bot_token_encrypted varchar(255)
  discord_notifications_enabled boolean [default: false]

  // Event policies
  allow_solution_history boolean [default: false] // FR-2.22, FR-3.20
  event_timezone varchar(50) [default: 'UTC']

  created_at      datetime [default: `now()`]
  updated_at      datetime

  indexes {
    status
    (start_time, end_time)
  }

  Note: "Single row table storing all runtime-configurable CTF settings"
}
```

```
////////////////////////////////////////////////////
// EVENT RULES (Content only)
////////////////////////////////////////////////////

Table event_rule_version {
  id              int [pk, increment]
  content_md      text [not null]
  version_number  int [not null]
  created_by      int [ref: > user.id, not null]
  created_at      datetime [default: `now()`]

  indexes {
    version_number
    created_by
    created_at
  }

  Note: "Stores all historical versions of event rules"
}

Table event_rule_current {
  id              int [pk]
  active_version_id int [unique, not null, ref: > event_rule_version.id]
  updated_at      datetime [default: `now()`]

  Note: "Points to the currently active rule version"
}

////////////////////////////////////////////////////
// NOTIFICATIONS (SIMPLIFIED - FORUM STYLE)
////////////////////////////////////////////////////

Table notification {
```

```
  id              int [pk, increment]
  title           varchar(100) [not null]
  message         text [not null]
   type               varchar(30) [not null] // 'announcement', 'event_update',
'system_alert'
  is_published    boolean [default: true]
  created_by      int [ref: > user.id, not null]
  created_at      datetime [default: `now()`]
  published_at    datetime

  indexes {
    type
    is_published
    (is_published, created_at)
    created_at
  }
}

//////////////////////////////////////////////////////
// AUDIT LOGGING
//////////////////////////////////////////////////////

Table audit_log {
  id              int [pk, increment]
  user_id         int [ref: > user.id]
   action           varchar(50) [not null] // 'challenge_solved', 'team_joined',
'flag_submitted'
  resource_type   varchar(30) // 'challenge', 'team', 'submission'
  resource_id     int
  details         json
  ip_address      varchar(45)
  created_at      datetime [default: `now()`]

  indexes {
```

```
    user_id
    action
    resource_type
    (resource_type, resource_id)
    created_at
  }
}
```