POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

CodeKataBattle
Design Document

Authors:

Gigante Davide (Personal code: 11018245)
Topazio Federica (Personal code: 10985760)

Academic Year: 2023-24

Contents

Contents **i**

1 Introduction **1**
1.1 Scope . . . 1
1.2 Definitions, Acronyms, Abbreviations . . . 3
1.2.1 Definitions . . . 5
1.2.2 Revision history . . . 6
1.2.3 Reference Documents . . . 6
1.3 Document Structure . . . 6

2 Architectural Design **9**
2.1 Overview: High-level components and their interaction . . . 9
2.2 Component View . . . 12
2.3 Deployment View . . . 15
2.4 Run Time View . . . 17
2.5 Component Interfaces . . . 33
2.5.1 Web Server Interfaces . . . 33
2.5.2 User Management Service Interfaces . . . 37
2.5.3 Email Service Provider Interfaces . . . 38
2.5.4 Tournament Management Service Interfaces . . . 39
2.5.5 Notification Service Interfaces . . . 40
2.5.6 Battle Management Service Interfaces . . . 41
2.5.7 Score Calculation Service Interfaces . . . 42
2.5.8 Static Analysis tools Interfaces . . . 43
2.5.9 GitHub Service Interfaces . . . 43
2.6 Selected Architectural Styles and Patterns . . . 44
2.6.1 Other Design Decisions . . . 45

3 User Interface Design **47**
3.1 Guest . . . 47
3.1.1 Home Page . . . 47
3.1.2 Sign up / Log in Interface . . . 48
3.2 Educator . . . 49
3.2.1 Sign Up Page . . . 49
3.2.2 Log In Page . . . 49
3.2.3 Home Page . . . 50
3.2.4 Create a New Tournament Interface . . . 50
3.2.5 Personal Area Page . . . 52
3.2.6 Change Settings Interface . . . 53
3.2.7 Notifications Page . . . 54
3.2.8 Contact Us Page . . . 55
3.2.9 Available Tournaments Page . . . 55
3.2.10 Tournament Page . . . 56
3.2.11 Create a New Battle Interface . . . 56
3.2.12 Battle Page . . . 58
3.2.13 Project Page . . . 59
3.3 Student . . . 60
3.3.1 Sign Up Page . . . 60
3.3.2 Log In Page . . . 60
3.3.3 Home Page . . . 61
3.3.4 Personal Area Page . . . 61
3.3.5 Change Settings Interface . . . 62
3.3.6 Notifications Page . . . 63
3.3.7 Contact Us Page . . . 63
3.3.8 Available Tournaments Page . . . 64
3.3.9 Tournament Page . . . 64
3.3.10 Battle Page . . . 65
3.3.11 Join Battle Interface . . . 67

4 Requirements traceability **69**

5 Implementation, Integration, Testing **77**
5.1 Implementation and Integration . . . 77
5.2 Practical Implementation Plan . . . 78
5.2.1 Initial Phase: Core Service Components . . . 78
5.2.2 Secondary Phase: Score Calculation and External Integrations . . . 78

5.2.3 Final Phase: Notification Service and Web Server 79
5.3 Testing Plan 79
5.3.1 Phase-Wise Testing Approach 79
5.3.2 Comprehensive Testing Methodologies 80
5.3.3 User Acceptance Testing (UAT) 80
5.3.4 Iterative Testing and Continuous Integration 81

6 Effort Spent **85**
6.0.1 Effort-Topazio 85
6.0.2 Effort-Gigante 85

List of Figures **87**

List of Tables **89**

1 Introduction

The Design Document (DD) aims to be the baseline for implementation activities and integration and evaluation thanks to the identification of the order of implementation and the integration strategy as well as the support of the verification and validation. The document also refines the plan and previous project estimations of size, cost, and schedule

1.1. Scope

Informatics is a multidisciplinary field that lies at the intersection of computer science, information technology, and various domains of knowledge, ranging from healthcare and business to social sciences and beyond. It revolves around the collection, storage, analysis, and dissemination of information to make informed decisions, solve problems, and enhance the efficiency and effectiveness of various processes.

In today's digital age, informatics has become an integral part of our lives, profoundly shaping how we communicate, work, learn, and access information. Its impact is felt in nearly every facet of modern society. One of the fundamental aspects of informatics is the utilization of programming languages to harness the power of computers for various applications.

Hence, **CodeKataBattle (CKB)** aims to be a platform that wants to help *students* how to reason and how to solve problems related to the paradigm of programming. This platform is based on the concept of **katas**. Katas are an essential component of traditional martial arts training and serve various purposes, such as developing technique, muscle memory, balance, and mental focus.

The idea is to use the same approach of katas, but applied to programming exercises:

- **Problem-Solving Skills**: Just as katas teach how to respond to specific martial arts scenarios, programming challenges and projects teach problem-solving skills. You analyze a problem, break it down into smaller parts, and apply your knowledge of programming to find a solution.

- **Repetition and practice**: Karate practitioners repeat katas to develop muscle memory and precision. Similarly, in programming, practice is crucial. Repeatedly working on coding exercises, projects, and challenges helps reinforce your understanding and improve your coding skills.

These soft skills are obtained through training with peers, challenging each other using **code kata battles**. A code kata battle is a coding exercise, that belongs to a specific *tournament*. When a tournament is created, the platform automatically notifies all the subscribed students, who must enroll in it before the specified subscription deadline, if they intend to participate. The students, who have chosen to participate in a tournament, will be notified for each battle published in it.

The *educator* who initiates the tournament has the authority to grant other instructors to create battles within it. When defining a battle, the educator must specify on CKB platform:

- a programming language of choice;
- a **code kata**, defined as a brief textual description and a software project, including test cases and build automation scripts;
- minimum and maximum number of students per group, formed by students themselves;
- a registration deadline;
- a final submission deadline;
- eventual additional configurations for scoring.

Once the battle registration deadline expires, CKB created a GitHub repository, which link is sent to all the subscribed groups of students. Students begin by forking the code kata's GitHub repository and setting up a GitHub Actions automated workflow to notify the CKB platform when they push new commits to the main branch. Each push triggers the CKB platform to update the team's battle score. The score, between 0 and 100, considers various automated factors:

- **functional aspects**, measured in terms of number of test cases that pass out of all test cases (the higher the better);
- **timeliness**, measured in terms of time passed between the registration deadline and the last commit (the lower the better);
- **quality level of the sources**, extracted through static analysis tools that consider

multiple aspects such as security, reliability, and maintainability (the higher the better). Aspects are selected by the educator at battle creation time

and optional manual evaluation factors:

- after the submission deadline, **personal score assigned by the educator**, who checks and evaluates the work done by students (the higher the better).

The platform continuously updates the battle scores as students push changes, allowing both students and educators to track the evolving rank. Once the consolidation is complete, students are notified of the final battle rank.

At the end of each battle, students' personal tournament scores are updated, reflecting their overall performance in that tournament. The students' overall performance is given by the sum of all battle scores, received during the tournament itself. This information is accessible to all platform users, showing ongoing tournaments and their corresponding ranks. Educators can close tournaments, and when the final rank is available, notifications are sent to all participating students. The CKBplatform is described as a microservices-based system, therefore as a distributed architecture with multiple, loosely-coupled services. This will thoroughly explained in more details in the following sections.

1.2. Definitions, Acronyms, Abbreviations

Abbreviations

- CKBplatform CodeKataBattle platform

Acronyms

Acronym	Meaning
RASD	Requirements Analysis and Specification Document
DD	Design Document
UML	Unified Modelling Language
WPX	World Phenomenon number X
SPX	Shared Phenomenon number X
GX	Goal number X
DX	Domain assumption number X
RX	Requirement number X
PRX	Performance Requirement number X
API	Application Programming Interface
MTTF	Mean Time To Failure
MTTR	Mean Time To Recovery
HTTP	Hypertext Transfer Protocol
XDR	eXternal Data Representation
SQL	Structured Query Language: it's a domain-specific languages used for managing data stored in relational database management systems.
SMPT	Simple Mail Transfer Protocol: it's the main protocol for transmitting emails.
TCP	Transmission Control Protocol: it's one of the main transportation layer protocols of the Internet Protocol suite, which provides a reliable (under normal conditions) delivery of information on the Internet.
REST	REpresentational State Transfer: it's a software architectural style and describes an interface for the communication between remote entities (this is also a nice way to describe the web).
OS	Operating System: it's the piece of software (of which the kernel is part) that manages hardware and software resources, providing utilities to developers.
IP	Internet Protocol: it's the main network layer protocol of the Internet.
HTTP	HyperText Transfer Protocol: it's one of the application layer protocols used in the Internet Protocol suite for distributing content and information through the network. Moreover, HTTPS represents the Secure version of HTTP (with data encryption), while HTTP3 stands for the third main version of the standard.

Table 1.1

1.2.1. Definitions

Users

- **User**: A generic User: can be either a guest, a student, or an educator.
- **Guest**: An Unregistered User who can register to CKB as either a student or an educator
- **Student**: A registered user, who typically engage in code kata battles, enroll in tournaments, and interact with their peers to practice and improve their programming skills.
- **Educator**: A registered user, who initiates a coding tournament, defines its parameters, has the authority to grant other instructors the ability to create coding battles within that tournament, and can create battles himself.

GitHub definitions

- **GitHub**: A web-based platform that provides version control and collaboration features for software development. In CKB, GitHub is used to host repositories for code katas and facilitate collaboration among students.
- **Commit**: A commit in the context of CKB refers to a specific snapshot or version of code changes made by students. These commits are pushed to the GitHub repository's main branch and are used to track progress and trigger automated scoring.
- Fork: To fork a GitHub repository means to create a personal copy of that repository under your GitHub account. In CKB, students fork the code kata's repository to work on their solutions independently.
- **Push**: Pushing code refers to the action of uploading code changes made by students to the main branch of the GitHub repository. When students push new commits, GitHub Actions are triggered to notify the CKB platform, and battle scores are updated accordingly.
- **GitHub Repository**: A digital space on the GitHub platform where code katas and related software projects are stored and managed. Each tournament's coding exercises have their own dedicated repositories.
- **GitHub Actions**: Automated workflows set up by students in their GitHub repositories to notify the CKB platform when they push new code commits to the main branch. This triggers the platform to update the team's battle scores and track

their progress.

- Main Branch: The primary branch in the GitHub repository where students push their code changes. Updates to the main branch trigger GitHub Actions and score updates in CKB.

Platform definitions

- **kata**: Japanese word describing detailed patterns of movements practiced either solo or in pairs. Katas are executed as a specified series of a variety of moves, with stepping and turning, while attempting to maintain perfect form.
- **code kata**: A brief textual description and a software project, including test cases and build automation scripts.
- **code kata battle**: A coding exercise created by an instructor for a specific tournament, that includes code kata, number of members per group, registration and final submission deadlines, and eventual additional configurations.

1.2.2. Revision history

... 07/01/24 1.0 first release

1.2.3. Reference Documents

- Specification Document: Assignment RDD A.Y. 2023-2024
- Prof. Matteo Camilli's Software Engineering course
- IEEE Standard on Requirement Engineering (ISO/IEC/IEEE 29148)
- `https://docs.github.com/en/get-started/quickstart/github-glossary`
- `https://en.wikipedia.org/wiki/Karate_kata`
- Github Actions documentation: `https://docs.github.com/en/actions`

1.3. Document Structure

1. **Introduction**: A general introduction to the purpose and scope of the project. It also contains the definitions, acronyms, and abbreviations used throughout the document to clarify its reading, as well as the revision history and the reference documents to better point out how the document has been developed.

2. **Architectural Design**: An overview of the high-level components and their interactions, with a focus on both static and dynamic view, helped by diagrams.
3. **User Interface Design**: A representation of how the User Interface will look like
4. **Requirements Traceability**: An explanation about how the requirements defined in the RASD map to the design elements defined in this document.
5. **Implementation, Integration and Test Plan** This provides an overview of the flow of implementation of the various components of the system, also providing a plan for integrating the various components and testing them all.
6. **Effort spent**: The amount of time spent by each team member for each section.

2 Architectural Design

2.1. Overview: High-level components and their interaction

This section is intended to explain in detail all the components used by the system in order to offer all the functionalities. The figure below shows a high-level description of the System's architecture. As mentioned in the introduction, the System is organized in a distributed architecture, describing a micro-services system. All the components are better explained in the following paragraph. The main aspects of the design are:

- **Client-Server architecture.** It consists of a presentation tier, an application and a data tier. The firsts one allow the user to communicate with the server and shows him through the UI the answers it receives back from the application tier. The second one receives requests from users, elaborates answers, changes the status communicating with the data tier and sends back the needed information to users. Finally, the data tier stores, updates, and handles the information.
- **Microservices** Functionalities of the system are divided into several services in order to reduce dependencies between modules and to increase scalability, availability and maintainability of the system.

The figure below represents a high-level description of the System's architecture. As mentioned in the introduction, the System is organized in a micro-services architecture.

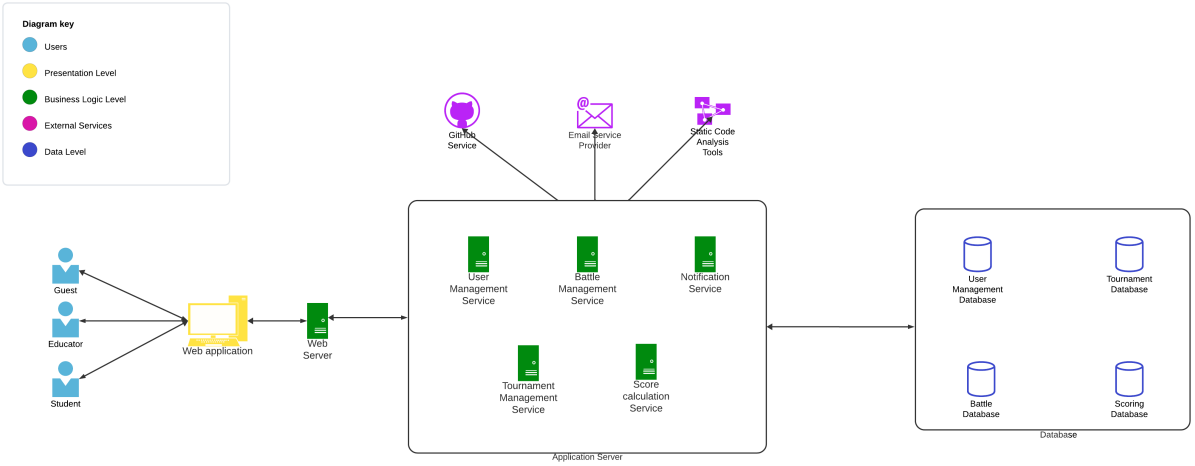

Figure 2.1: High-level description of the System's architecture

Categorization of the components of the distributed system provided through an high-level view of the system's architecture.

- **Users** The CKBplatform end-users who use their devices to interface with the System
  - **Guest** The Unregistered User who interacts with the CKBplatform (web application) through his personal computer
  - **Educator** The Registered User who interacts with the CKBplatform (web application) through his personal computer
  - **Student** The Registered User who interacts with the CKBplatform (web application) through his personal computer
- **Presentation Level** The user interface and communication level of the CKB system where end-users (Students, Educators, and Guests) interact with the application.
  - **Web Application** A web-based interface accessible through users' browsers, enabling interaction with the CKB platform's functionalities. It communicates with the system by sending requests to the micro-services. This application provides a user-friendly interface for managing tournaments, viewing scores, available battles, available tournaments, and different kinds of enrolments.
- **Web Server** back-end component of the Web Application that communicates with the User's browser and with the Application Server(Business Logic).

- **Business Logic** This level is composed of various micro-services, each handling a specific piece of the business logic. Despite being distributed, they collectively form the business logic layer of the system.
  - **User Management Service** Handles all user-related operations, including registration, authentication, profile management, and notifications settings for Students and Educators. Essentially, this service is responsible for account creation when the users sign up and for authentication when the users log in
  - **Tournament Management Service** Responsible for the creation, modification, and closure of tournaments. Manages tournament metadata, rules, and educator permissions.
  - **Battle Management Service** Facilitates the creation and management of battles within tournaments. Includes setting up battles, managing deadlines, and updating battle statuses.
  - **Score calculation Service** Computes scores based on automated evaluations (like test case results) and manual evaluations done by educators. Integrates with external static analysis tools for code quality checks.
  - **Notification Service** Manages the sending of notifications and alerts to users about tournament and battle updates and score availability.
- **Data Level** The level whose resources are accessible only to the business logic level and contains the data storage which is responsible for the storage of all the CKB system's data, that with a micro-services architecture should be decentralized.
  - **User Management Database** Stores user profiles, credentials, preferences, and notification settings.
  - **Tournament Database** Contains information about tournaments, including metadata, participating educators, configuration settings, and information about it.
  - **Battle Database** Holds details about battles, including code katas, participant lists, and group formations, and related links and metadata related to code repositories.
  - **Scoring Database** Records scores, test results, and manual evaluations for battles and tournaments.
- **External Systems** Systems whose functionalities are integrated with the CKB

system but developed and maintained by third parties.

- **GitHub System** Manages the code submission and retrieval process for the CKB system, providing a platform for version control and collaborative coding.
- **Email Service Provider** Handles the delivery of automated emails from the CKB system to its users. This system is crucial for communications regarding tournament updates, battle invitations, new tournaments and battle notifications, and other alerts.
- **Static Code Analysis Tools** Third-party services integrated into the CKB system for analyzing the quality of code submissions. These tools provide metrics used in score calculation, based on the quality aspects with respect to which the code is evaluated.

2.2. Component View

In this Section, the System's high-level components are shown and analyzed. The most complex ones are further detailed with their sub-components specifying their functionalities. The color code used in the component diagram is the same used in the Overview 2.1.

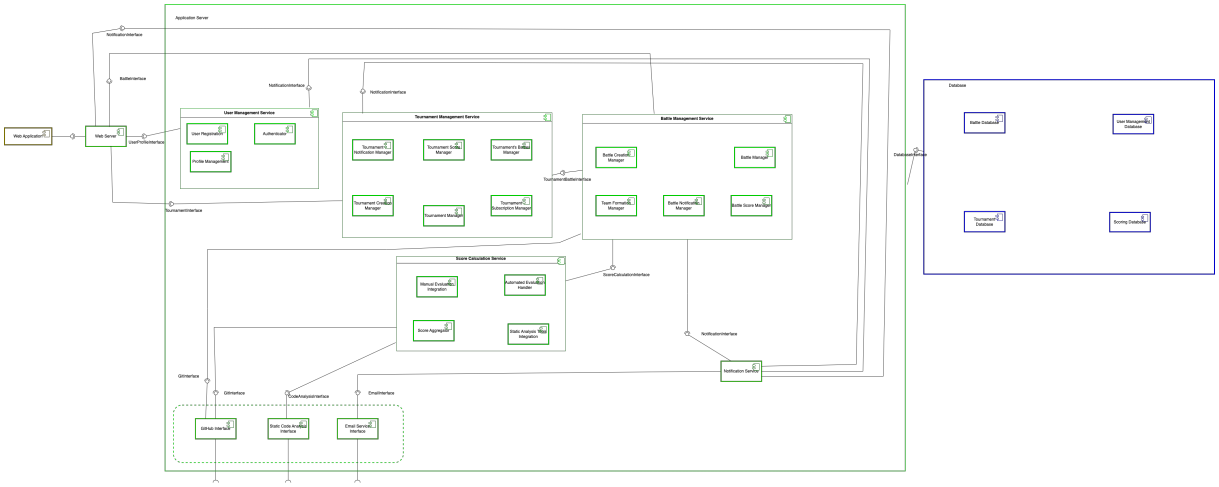

Figure 2.2: Component view of the System

- **User Interface Component** This is the front-end of the System through which the users interact with the CKB system. The only logic present in such components is the ability to perform basic checks, while all the computational tasks are carried out by the servers. For this reason, these components must be able to communicate. In particular, the User Interface Component is

  - **Web Application** The graphical user interface for users to interact with the system.

- **Web Server** receives HTTPS requests from the User's browser and forwards them to the application server to collect all the data required to generate the dynamic web pages

- **Business Logic Components**
  - **User Management Service** has 3 sub-components:
    * **User Registration**, responsible for new users sign-ups
    * **Profile Management**, allows users to access, manage, and update their personal profile information
    * **Authenticator**, handles the authentication by checking the users' credentials
  - **Tournament Management Service** has 6 sub-components:
    * **Tournament Notification Manager**, handles all kinds of notifications raising within the tournament setting.
    * **Tournament Creation Manager**, manages the creation of a new tournament
    * **Tournament Manager** responsible for the management of the ongoing tournament, deadline expiration, tournament state, closing of a tournament.
    * **Tournament Subscription Manager**, manages Students subscribed to that tournament.
    * **Tournament's Battles Manager**, interfaces with all the battles created within a specific tournament.
    * **Tournament Score Manager**, manages ranks, scores of Students subscribed within a tournament. Therefore, interfacing with the Score Calculation Service.
  - **Battle Management Service** has 5 sub-components:
    * **Battle Notification Manager**, handles all kinds of notifications raising within the battle setting.

    * **Battle Creation Manager**, manages the creation of a new battle
    * **Battle Manager** responsible for the management of the ongoing battle, deadline expiration's dates, battle state.
    * **Team Formation Manager**, manages the formation of teams within a battle context.
    * **Battle Score Manager**, manages ranks, scores of Teams participating in a battle. Therefore, interfacing with the Score Calculation Service.
  - **Score Calculation Service** has 3 sub-components:
    * **Automated Evaluation Handler**, applies automated scoring algorithms based on predefined criteria.
    * **Manual Evaluation Integration**, facilitates the inclusion of scores given manually by educators.
    * **Static Analysis Tools Integration**, facilitates the inclusion of scores obtained through the Static Analysis Tools, computing the quality metrics on the specified parameters.
    * **Score Aggregator**, aggregates all the computed scores.
  - **Notification Service**, manages the dispatch of various notifications and alerts to users regarding tournaments, battles, and scores.
- **Databases**:
  - **User Management Database**, stores all data related to users of the CKB platform.
  - **Tournament Database**, holds information related to the tournaments.
  - **Battle Database**, stores details concerning individual battles within tournaments.
  - **Scoring Database**, responsible for recording scoring and evaluation data.
- **External services interface** The system has 3 interfaces components that are in charge of communicating with the external services API. The interfaces are:
  - **GitHub Interface** Handles interactions with GitHub for repository management, code submissions, and leveraging GitHub Actions.

- **Email Service Interface** Manages communication with an email service provider for sending out notifications.
- **Static Code Analysis Interface** Integrates with third-party static code analysis tools, managing the analysis process and retrieving results.

2.3. Deployment View

This section illustrates the distribution of components capturing the topology of the System by using a deployment diagram. The deployment diagram will follow the same color coding as in the previous sections for consistency.

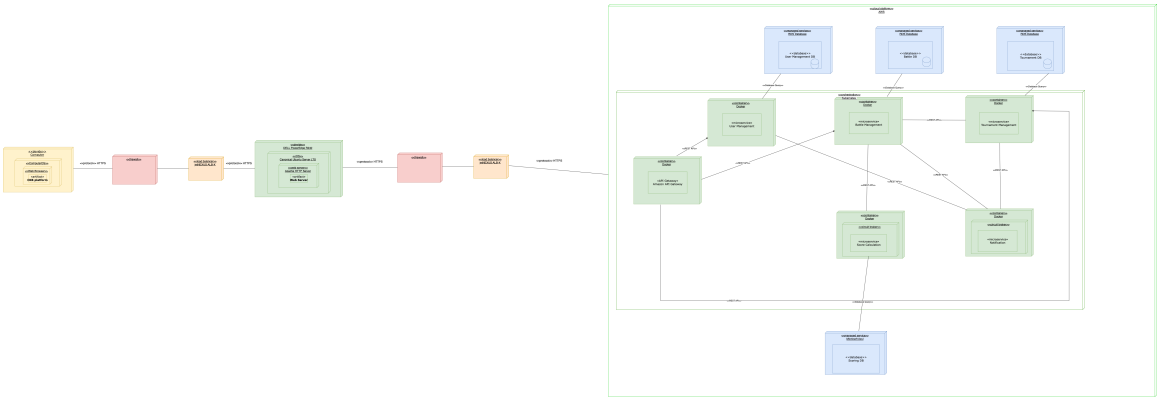

Figure 2.3: Deployment View

- **Client Devices**, personal computers or mobile devices used by the Users (Guests, Educators, Students) to interact with the CKB platform through the Web Application.
- **Web Servers**, servers that host the web application, processing HTTPS requests from users' browsers. They act as the intermediary between the client devices and the business logic layer.
- **Load Balancer**, ensures that the incoming requests are distributed efficiently across multiple servers, both before the Web Server, and the Application Servers. This not only optimizes resource utilization but also enhances the system's responsiveness and availability.
- **Firewall**, a network security device that monitors incoming and outgoing network traffic for security purposes. A firewall filters the traffic coming from the User's device and another one filters the traffic coming from the Web Server.

- **Cloud Platform**, hosts and manages all the components of the system, providing infrastructure services like compute instances, managed databases, and networking.
- **Orchestration Platform**, orchestrates and manages Docker containers in which the microservices run. It handles the deployment, scaling, and operational aspects of containers, and it provides a service discovery, which is a typical feature provided by the orchestration platform itself.
- **Docker Containers**, each microservice runs in its own Docker container, ensuring isolation, consistency, and portability.
- **API Gateway**, manages incoming API requests, routes them to the appropriate microservices, and handles responses. It also deals with cross-cutting concerns like authentication, rate limiting, and request transformation.
- **Application Servers**, these servers host the various microservices that constitute the Business Logic of the system. Each microservice is deployed independently, allowing for scalability and flexibility.
- **Circuit Breaker**, implemented within the service that makes external calls to other services, such as GitHub service, Notification Service and Score Calculation Service.
- **Managed DB**, provide data persistence and management services for the microservices. Each microservice typically communicates with its own database, whithin the cloud platform.

2.4. Run Time View

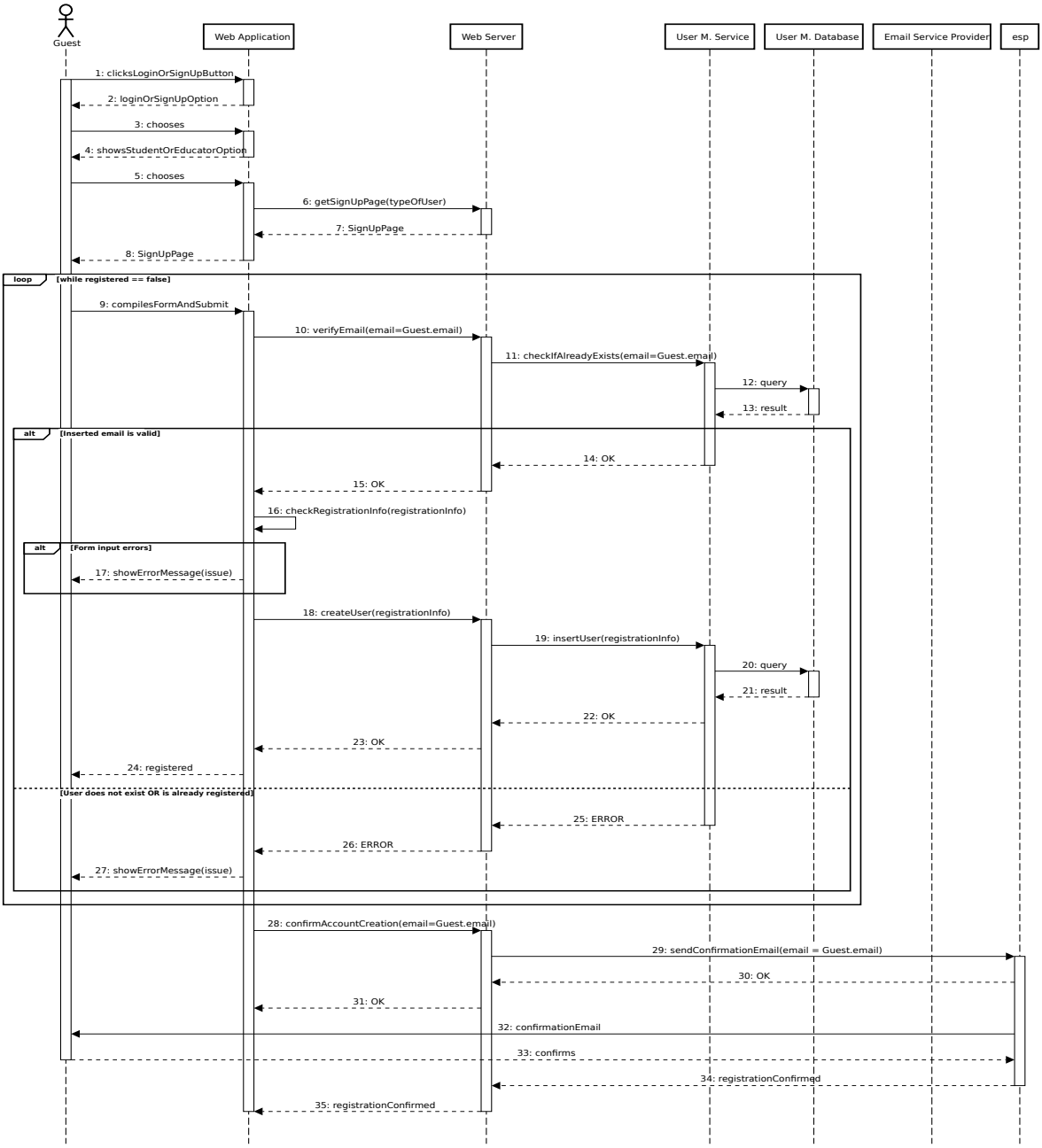

Figure 2.4: User Registration

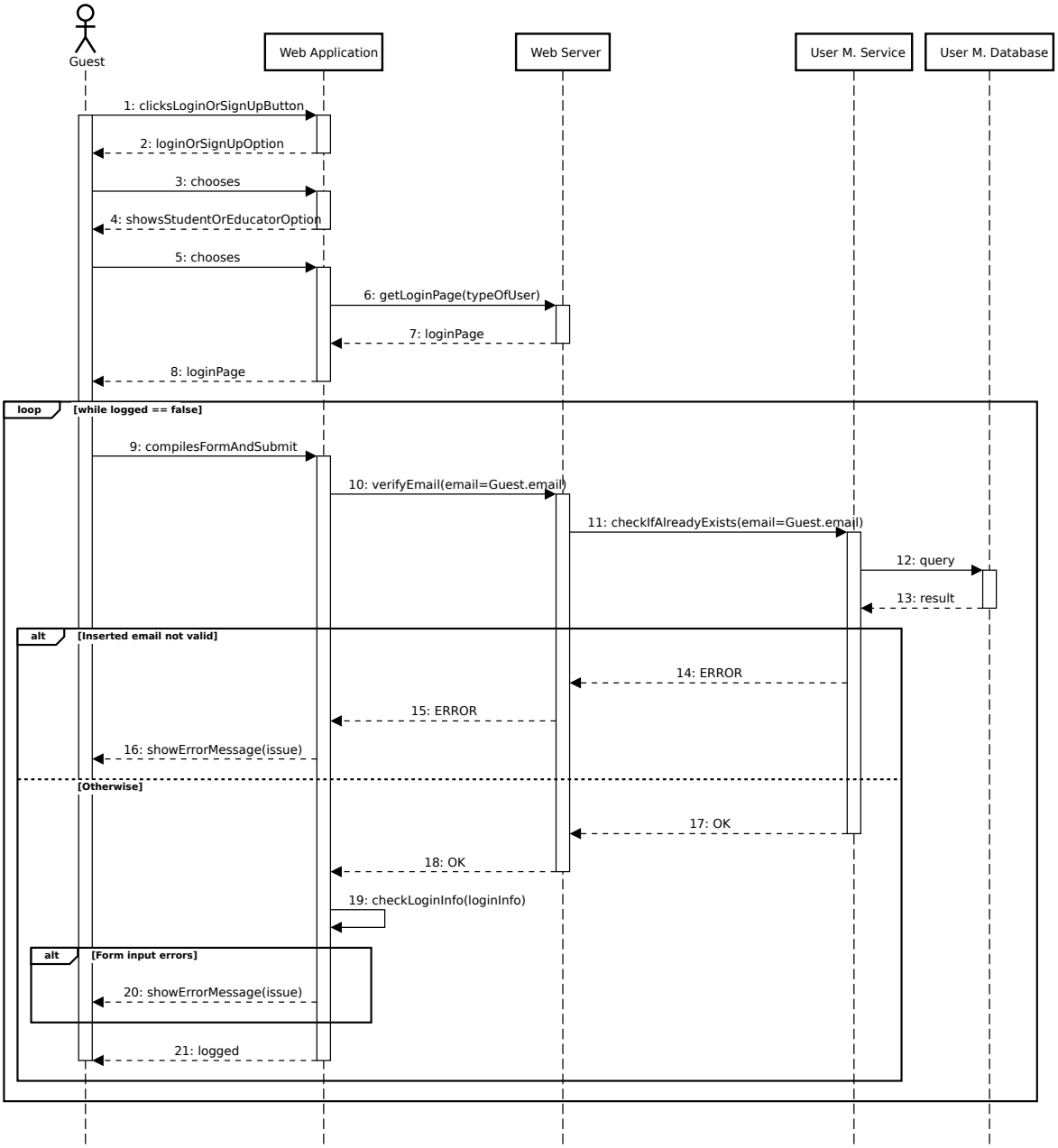

Figure 2.5: User Login

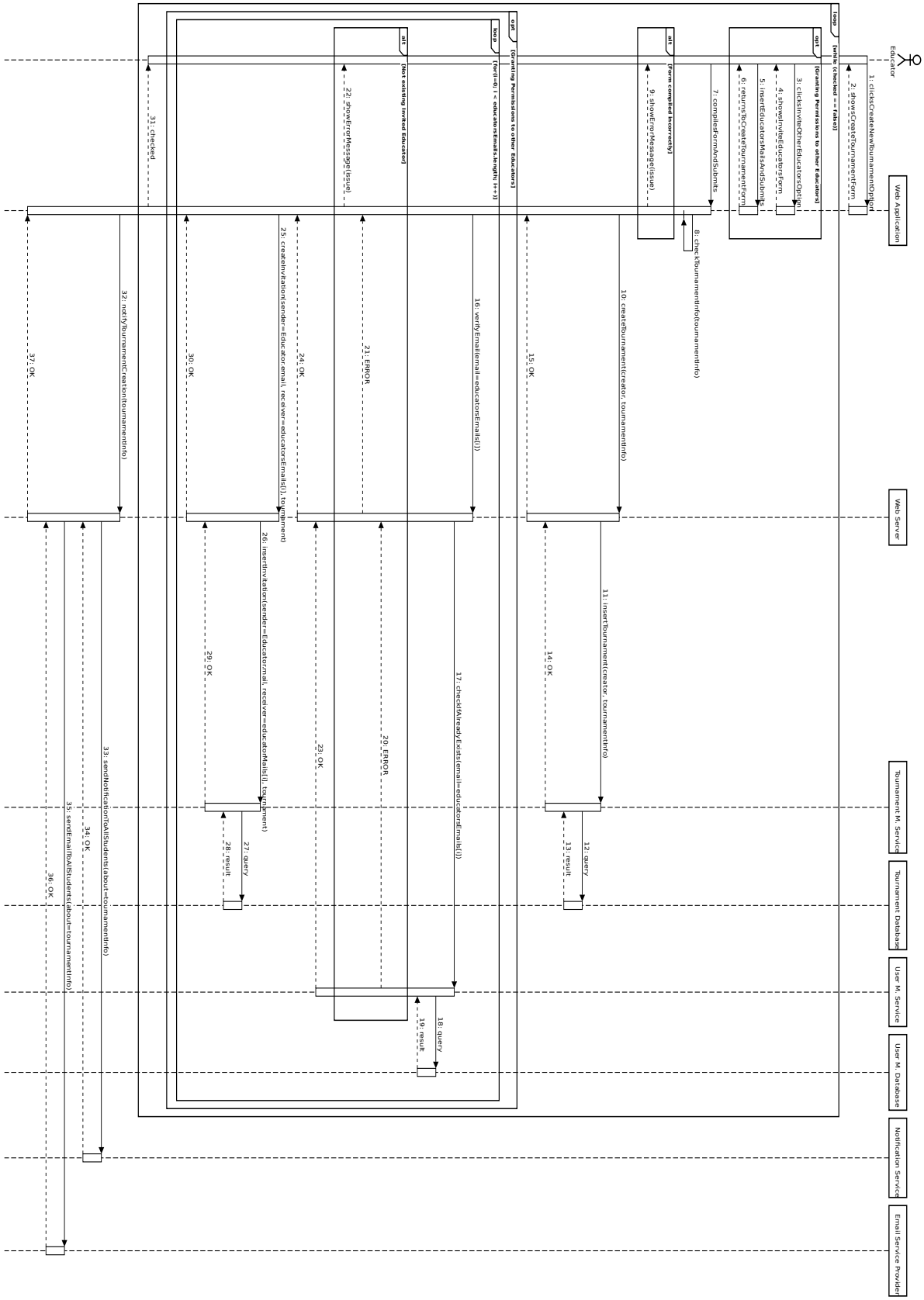

Figure 2.6: Tournament Creation

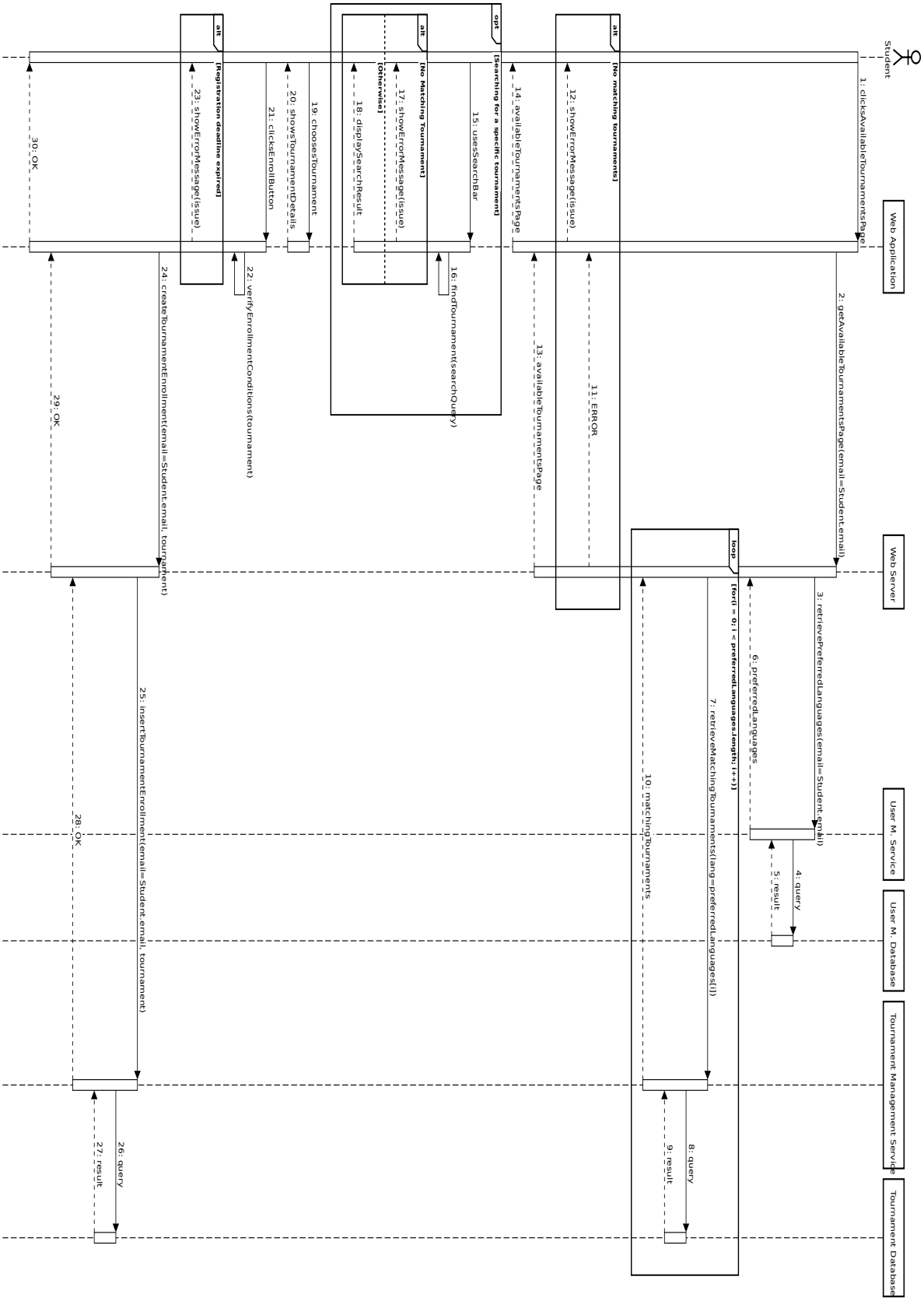

Figure 2.7: Tournament Enrollment

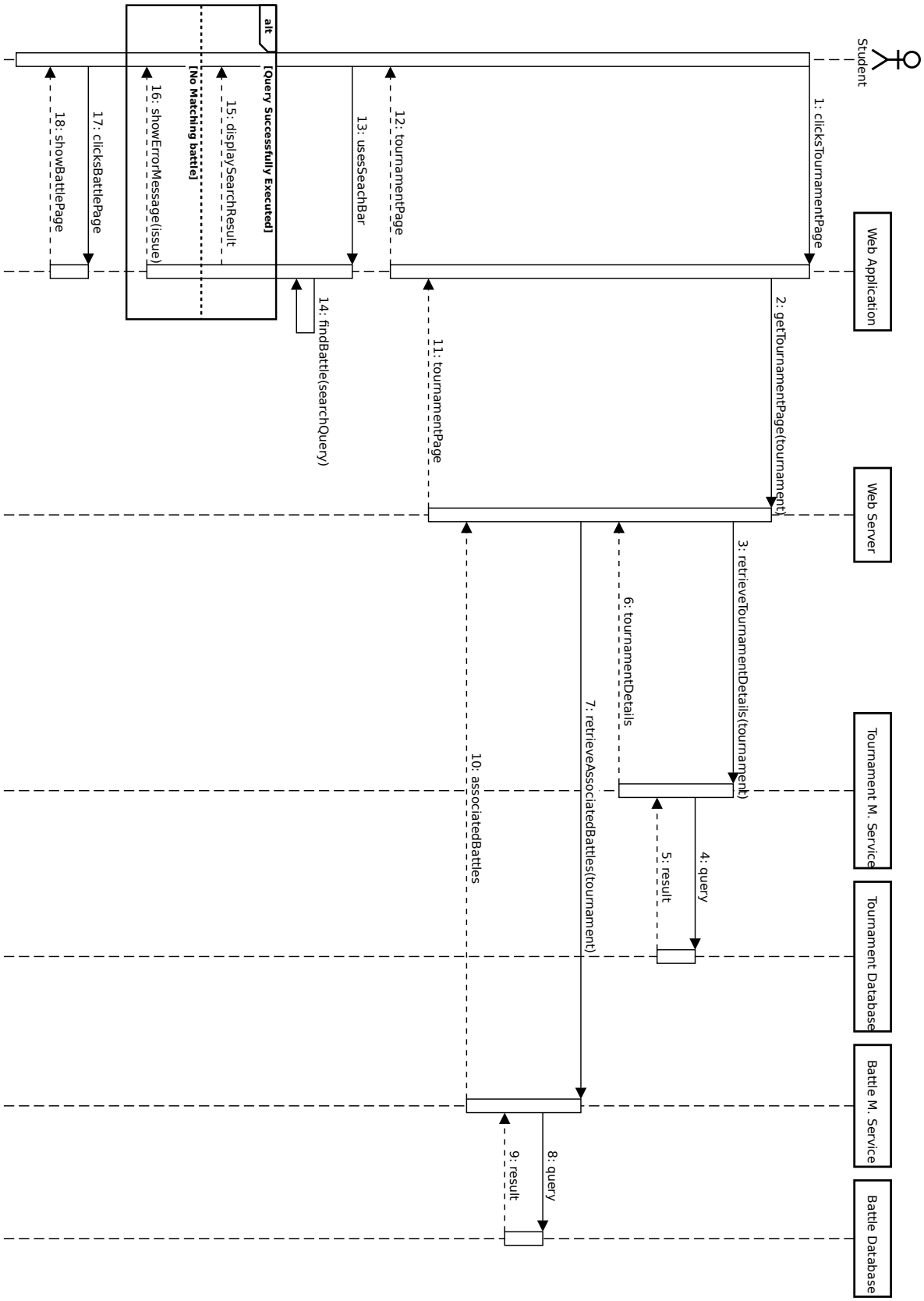

Figure 2.8: Battle search

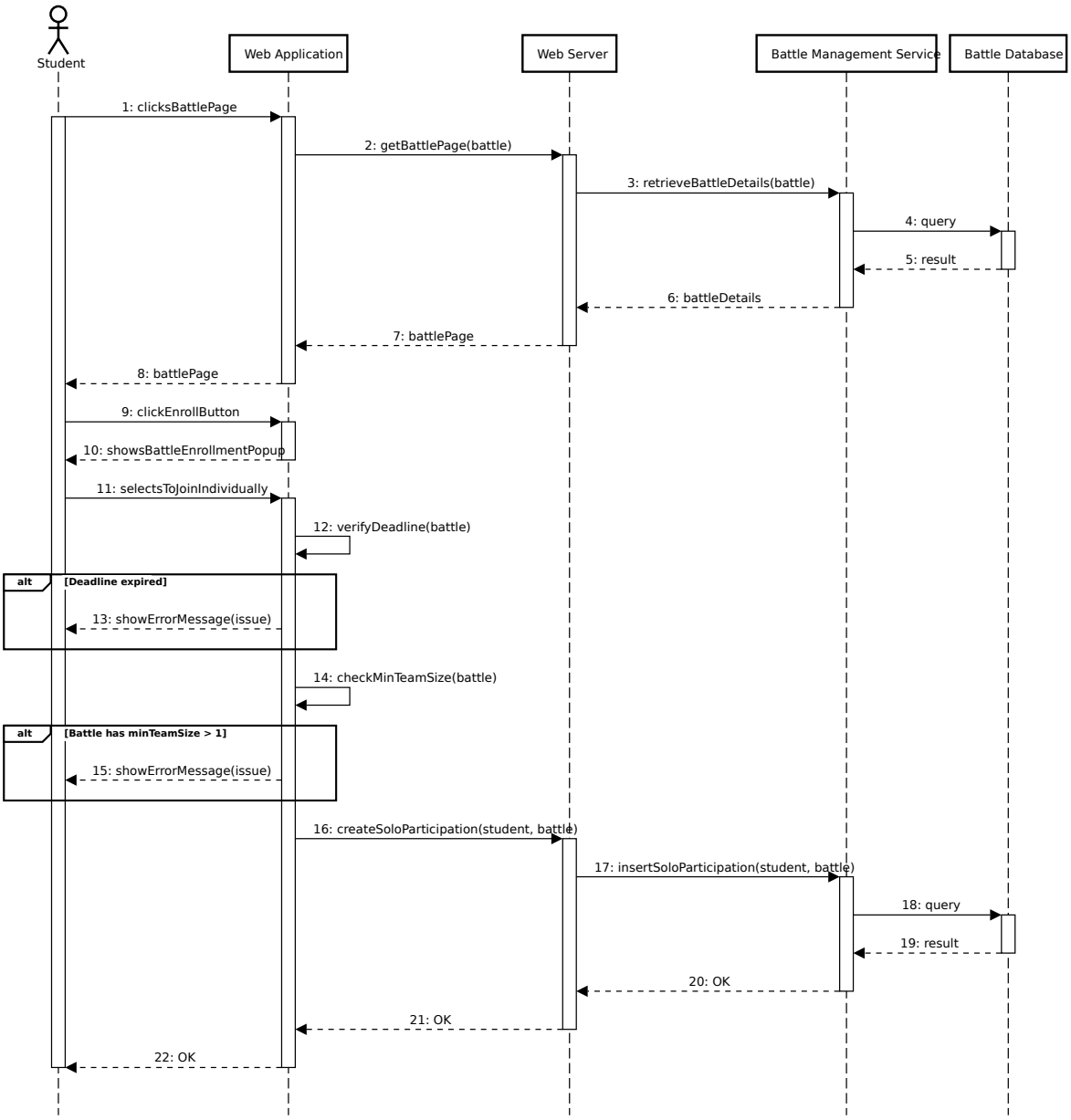

Figure 2.9: Join Battle Individually

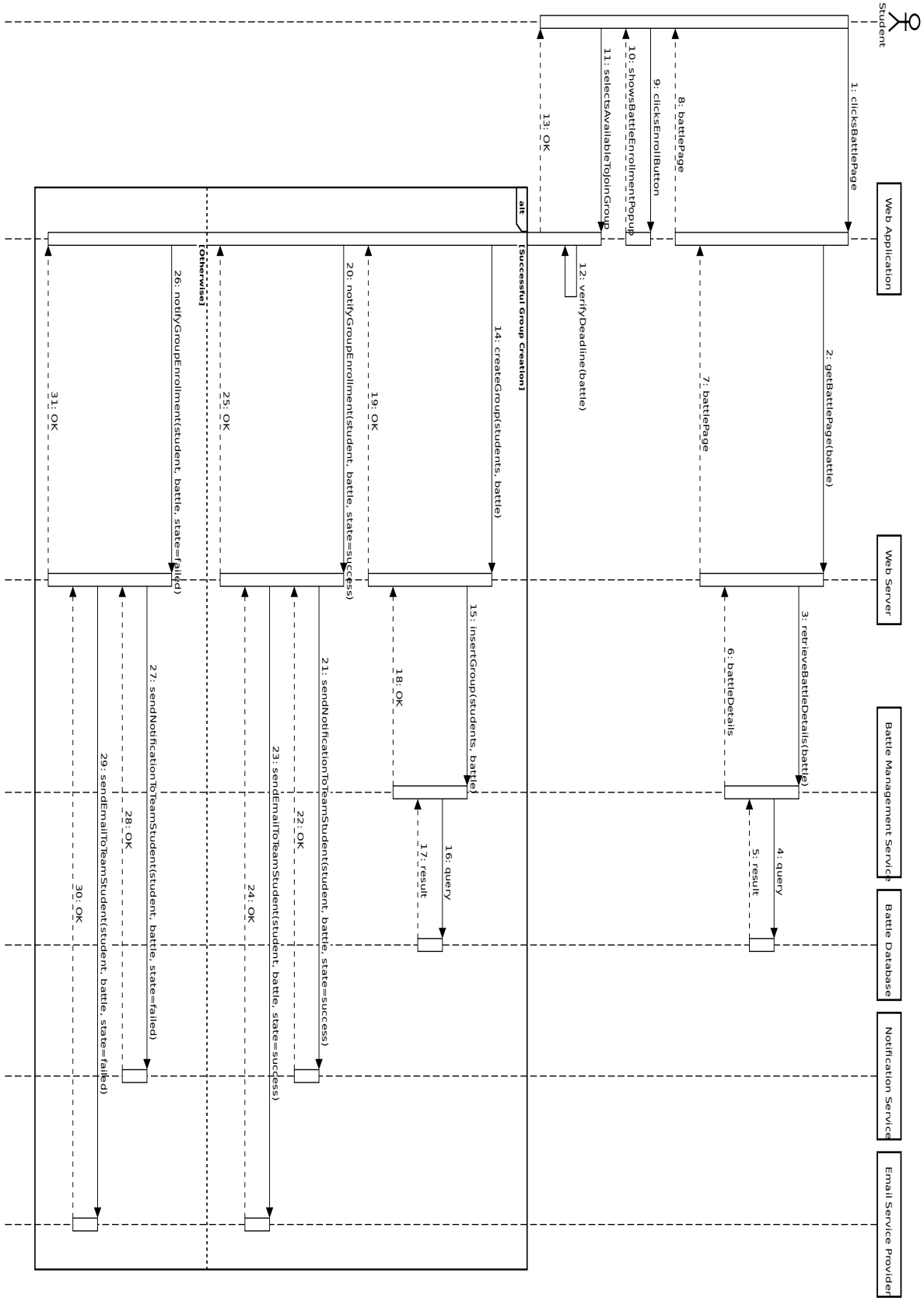

Figure 2.10: Group Formation

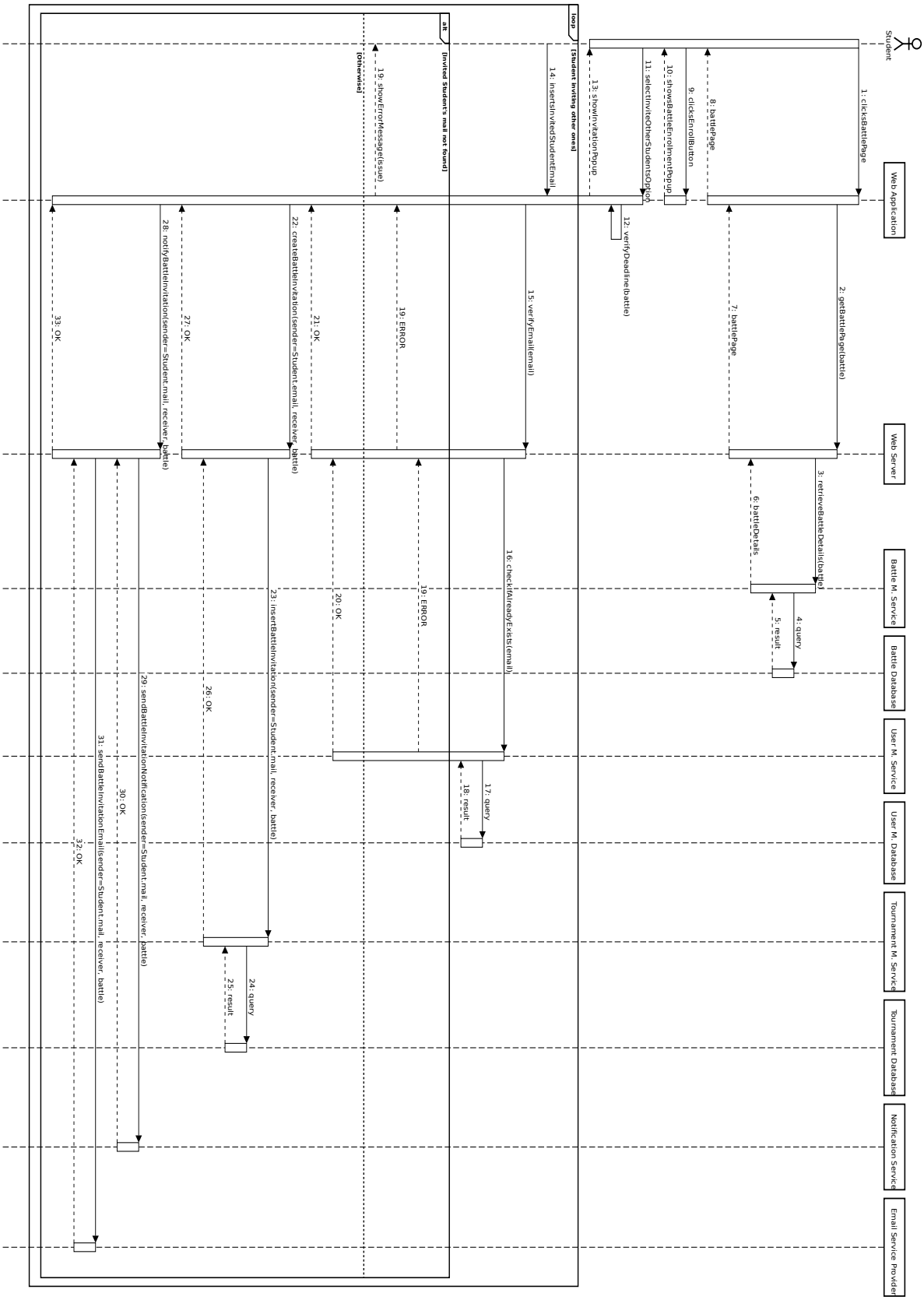

Figure 2.11: Send Battle Invitation

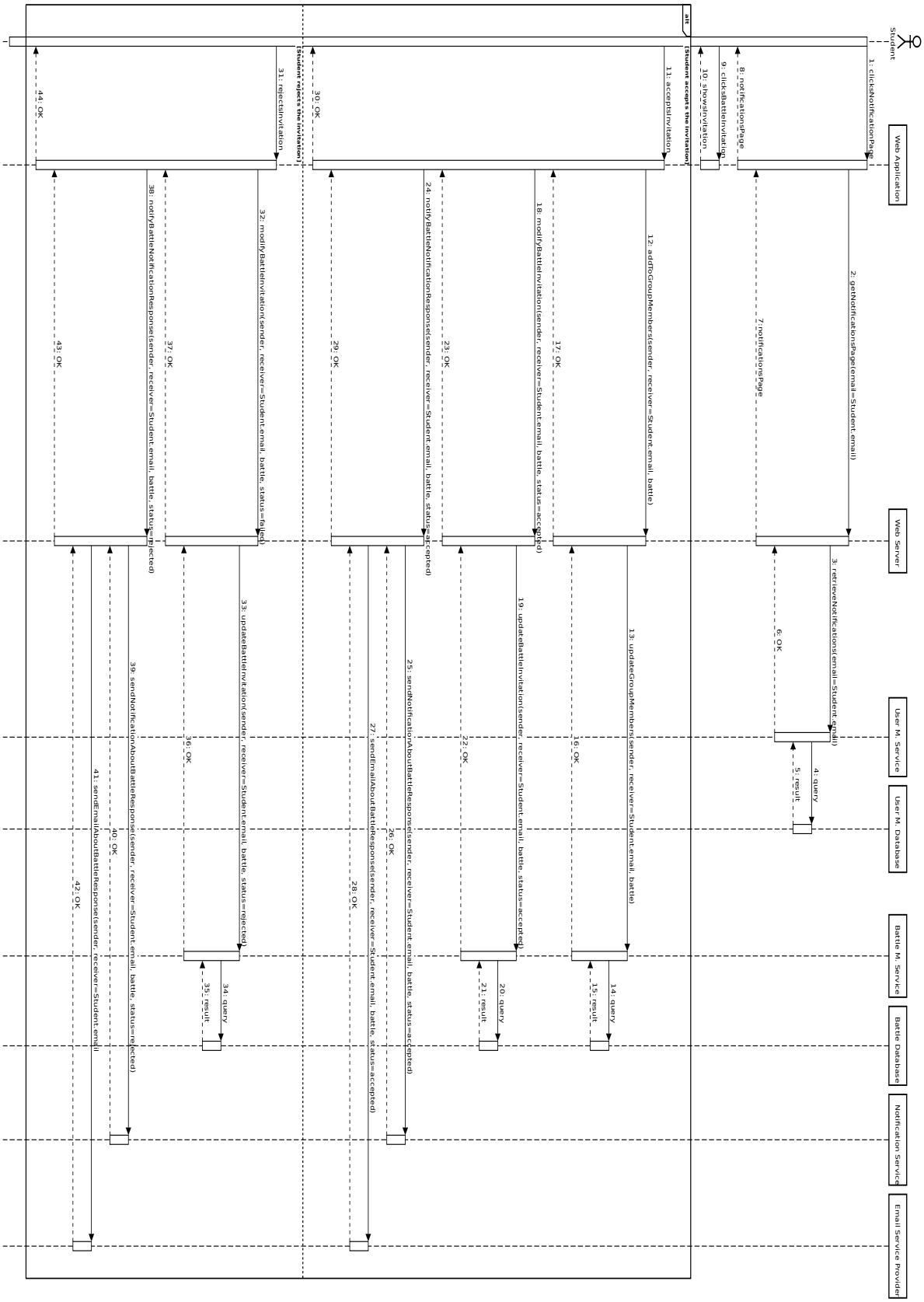

Figure 2.12: Join Battle By Invitation

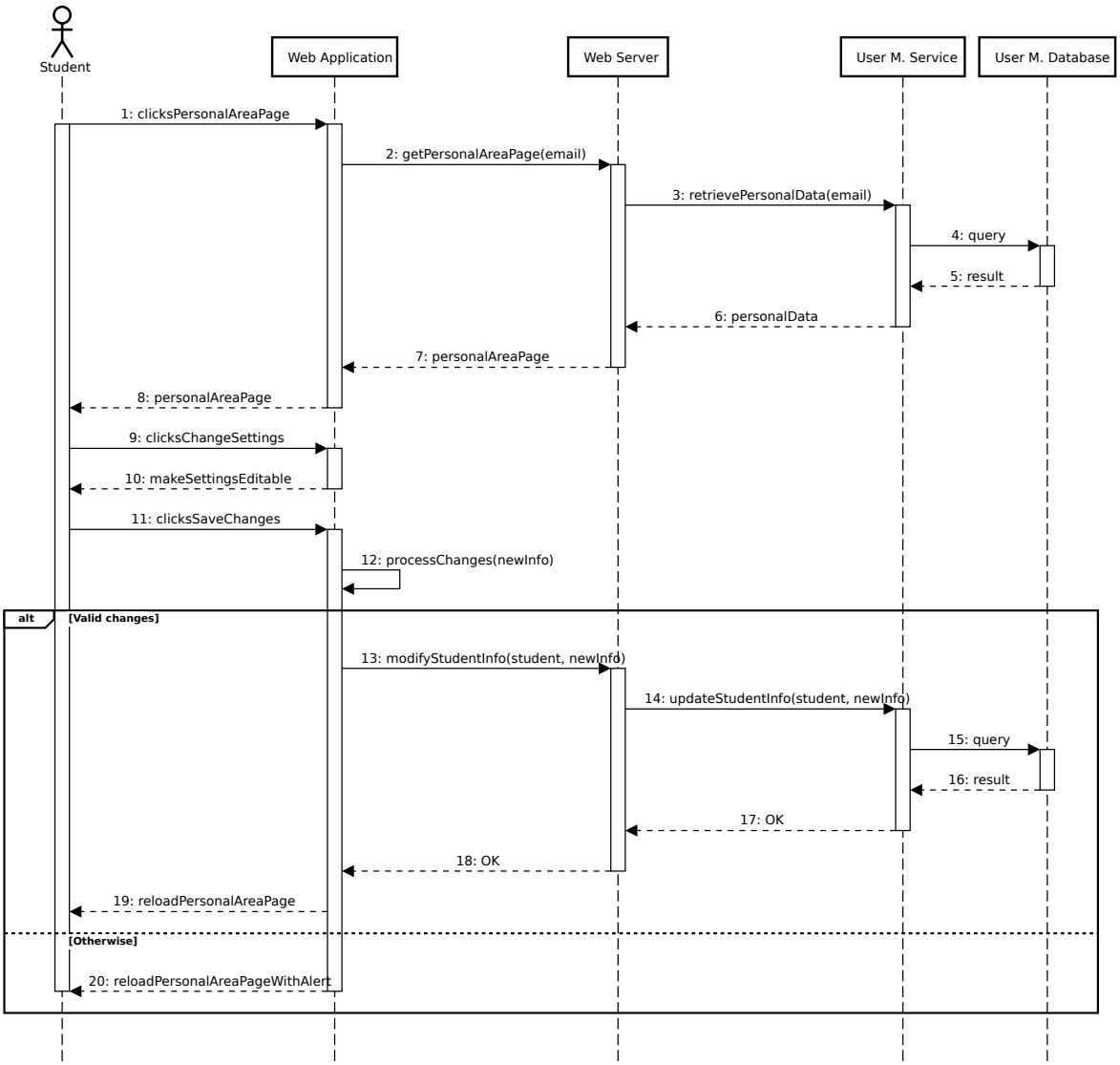

Figure 2.13: Change Languages Settings

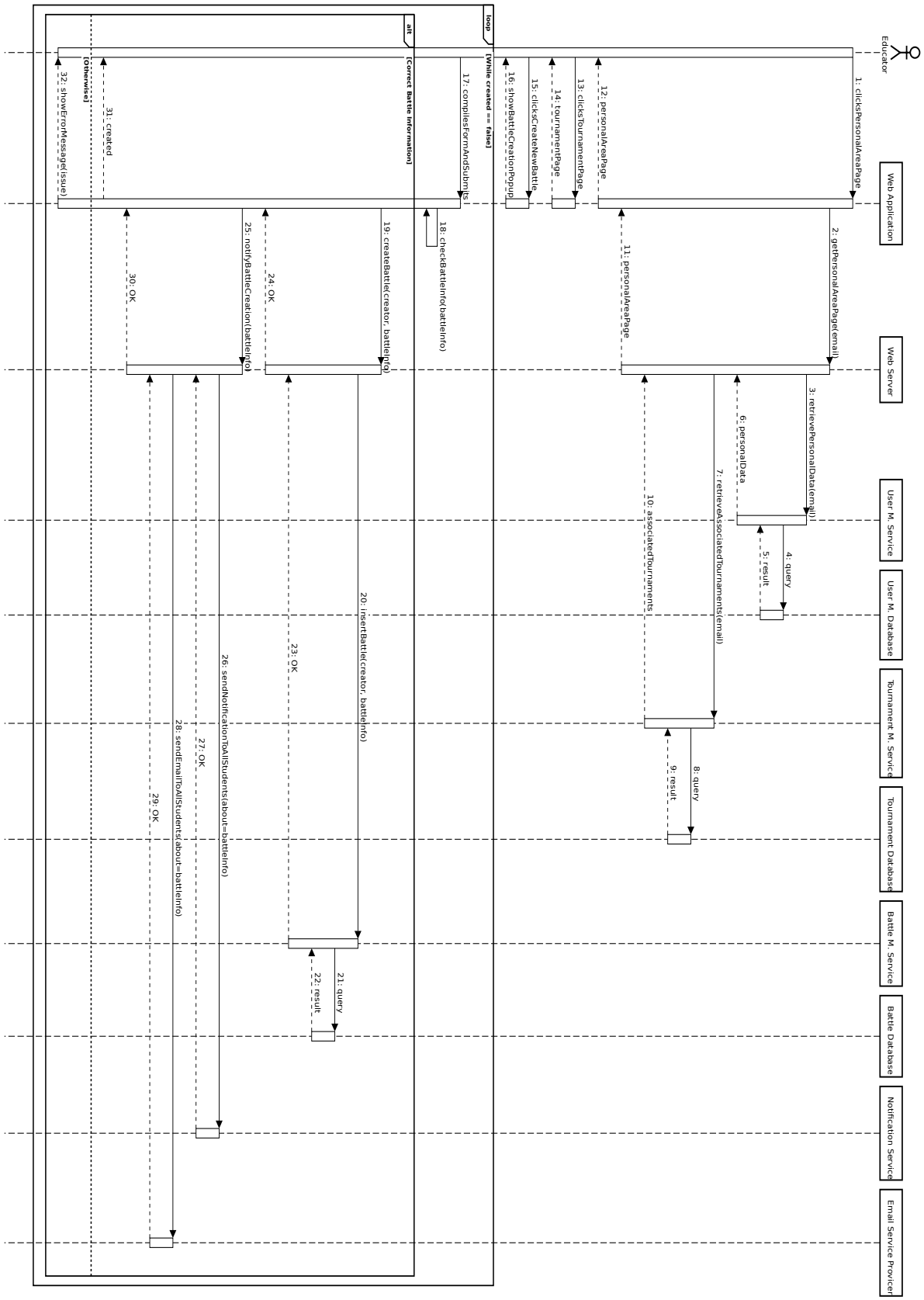

Figure 2.14: Battle Creation

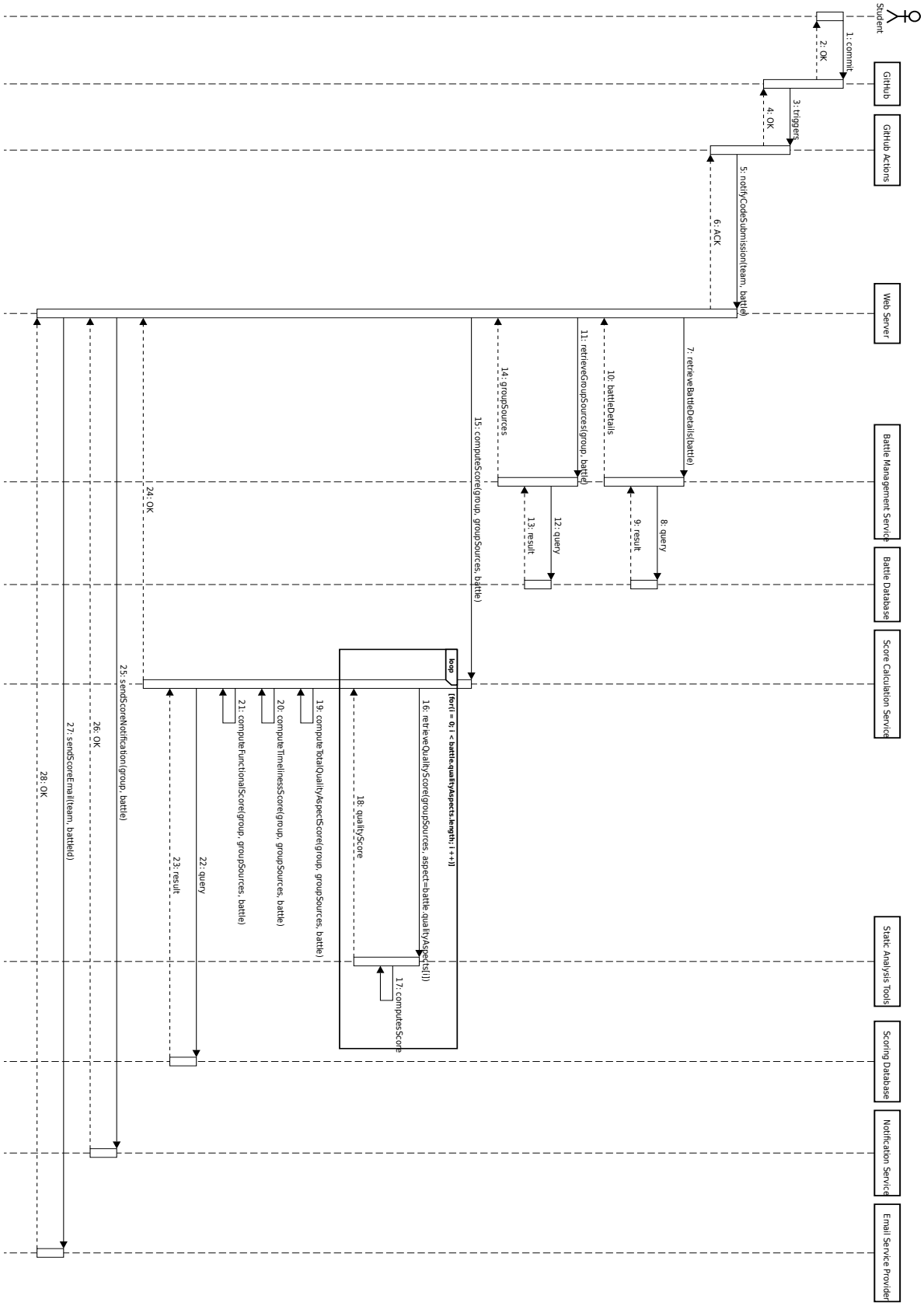

Figure 2.15: Score & Battle Update

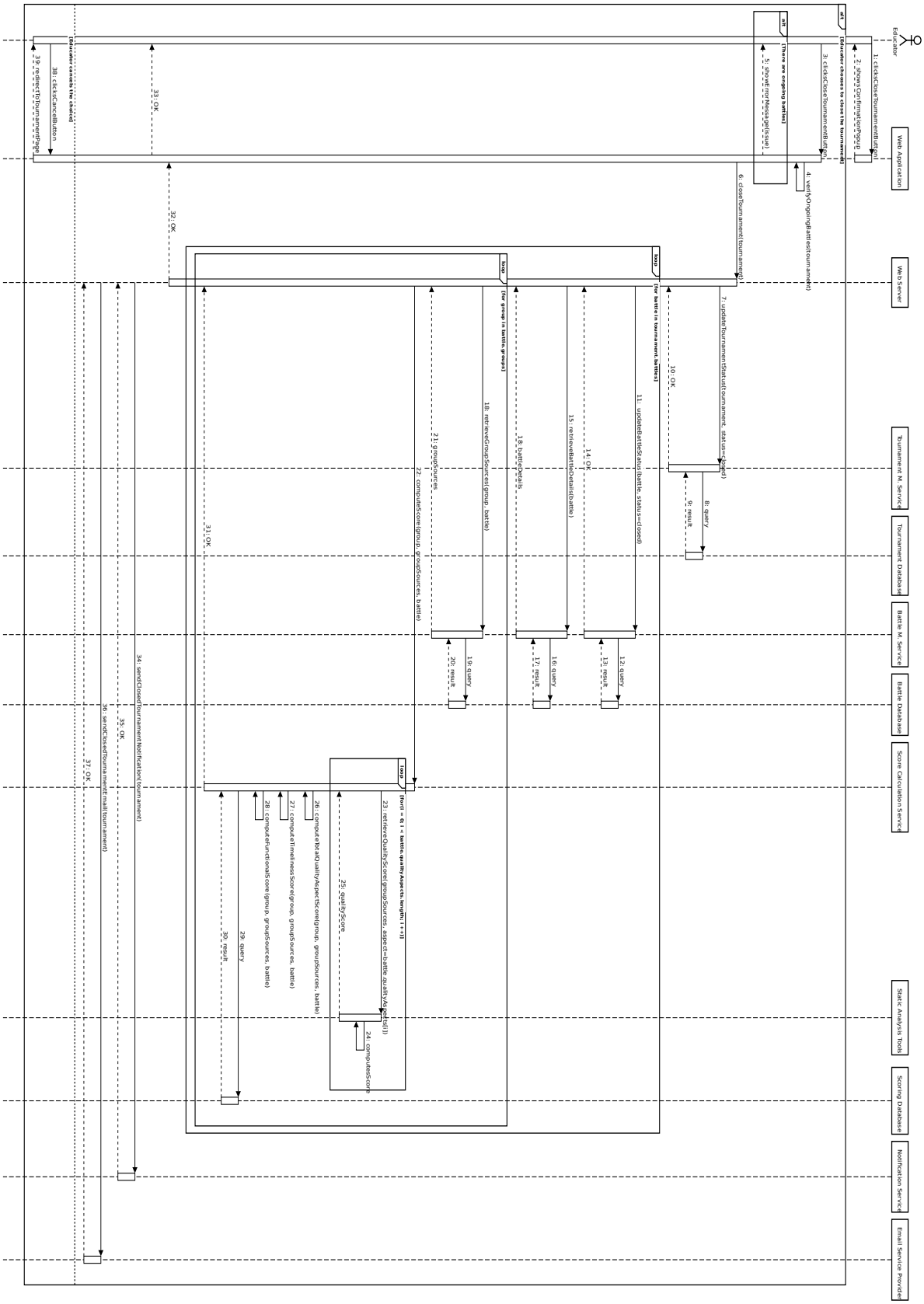

Figure 2.16: Close Tournament

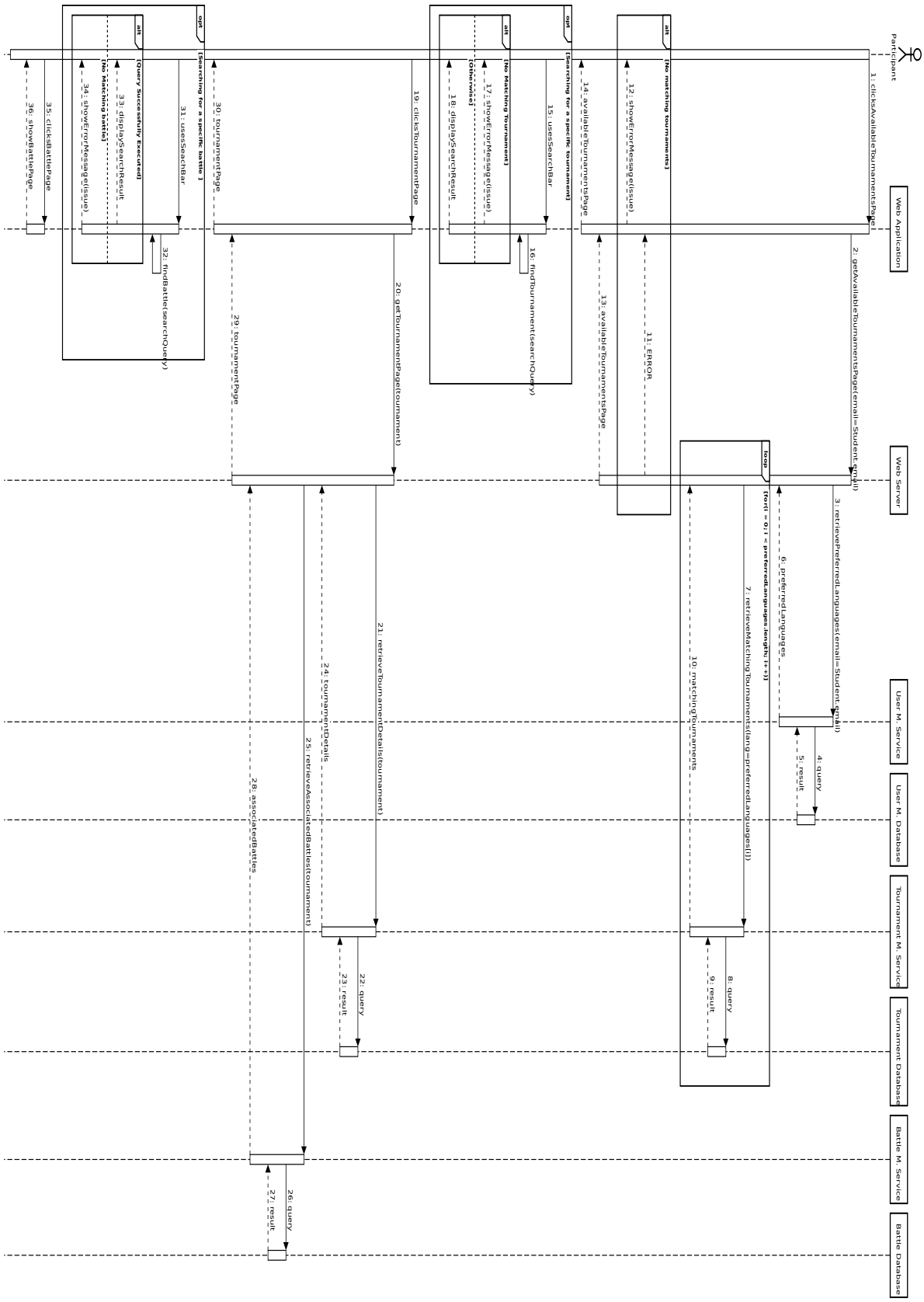

Figure 2.17: View Battle Score

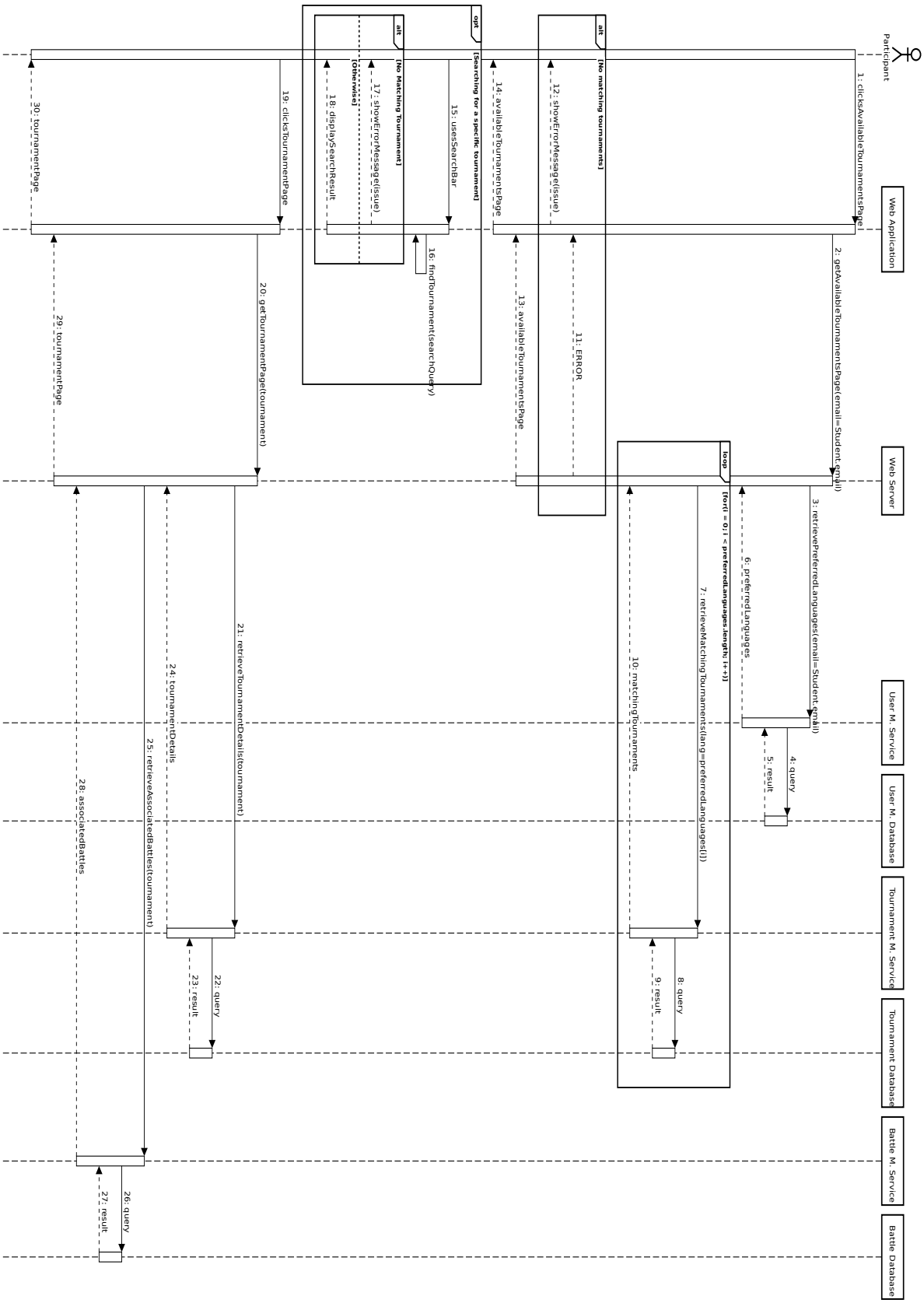

Figure 2.18: View Tournament Score

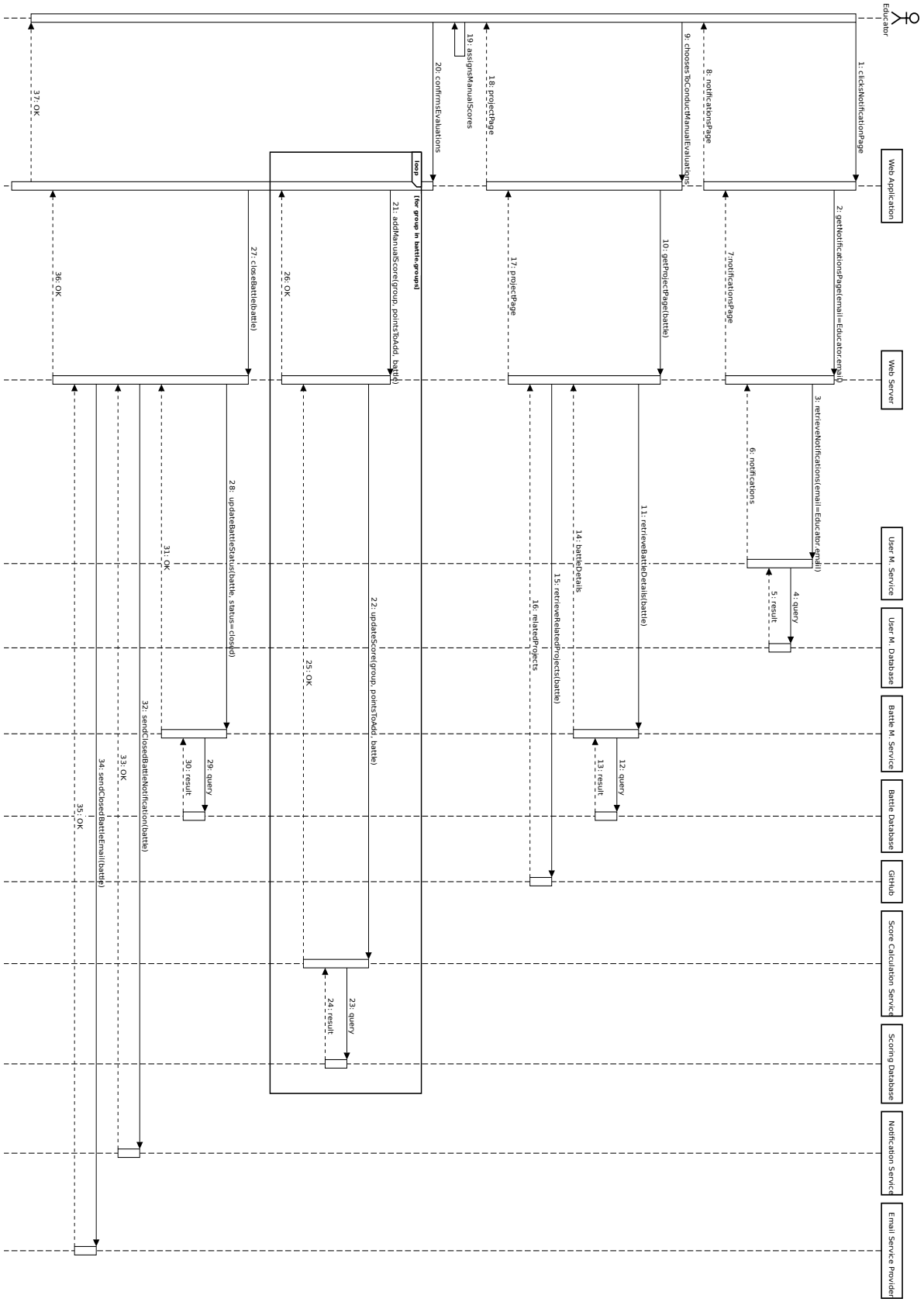

Figure 2.19: Manual Evaluation

2.5. Component Interfaces

2.5.1. Web Server Interfaces

- **getSignUpPage(typeOfUser)** Retrieves the sign-up page based on the type of user.
  - *typeOfUser*: Specifies the guest's choice between Educator and Student.
- **verifyEmail(email)** Verifies that the inserted email is correct.
  - *email*: Email to verify.
- **createUser(registrationInfo)** Submits the registration information to create the new User.
  - *registrationInfo*: A set of input parameters referred to the submit form, as name, email, etc.
- **confirmAccountCreation(email)** Starts the confirmation process of the newly created account.
  - *email*: Email address to which the confirmation email will be sent.
- **getLoginPage(typeOfUser)** Retrieves the log-in page based on the type of user.
  - *typeOfUser*: Specifies the guest's choice between Educator and Student.
- **createTournament(creator, tournamentInfo)** Submits the information to create the new Tournament.
  - *creator*: Tournament creator's email.
  - *tournamentInfo*: A set of input parameters referred to the tournament attributes, as minimum and maximum number of participants per team, etc.
- **createInvitation(sender, receiver, tournament)** Submits the information to create a new tournament help invitation, saving it with status *PENDING*.
  - *sender*: Tournament's creator email.
  - *receiver*: Invited educator's email.
  - *tournament*: The specific tournament of interest.
- **notifyTournamentCreation(tournamentInfo)** Starts the notification process of the newly created tournament.

  - *tournamentInfo*: A set of input parameters referred to the tournament attributes, as registration deadline, etc.

- **getAvailableTournamentsPage(email)** Retrieves the available tournaments page based on user's email.
  - *email*: User's email.
- **createTournamentEnrollment(email, tournament)** Submits the information to create the new tournament enrollment.
  - *email*: Enrolled student's email.
  - *tournament*: The specific tournament of interest.
- **getTournamentPage(tournament)** Retrieves the tournament page related to a specific tournament.
  - *tournament*: The specific tournament of interest.
- **getBattlePage(battle)** Retrieves the battle page related to a specific battle.
  - *battle*: The specific battle of interest.
- **createSoloParticipation(student, battle)** Submits the information to create a new individual battle participation.
  - *student*: Student's email.
  - *battle*: The specific battle of interest.
- **createGroup(students, battle)** Submits the information to create a new group of students within the context of a specific battle.
  - *students*: Array of students that have joined the group.
  - *battle*: The specific battle of interest.
- **notifyGroupEnrollment(student, battle, state)** Initiates the notification process for a student regarding a particular battle.
  - *student*: Student's email.
  - *battle*: The specific battle of interest.
  - *state*: The enrollment status, which can be either *SUCCESS* or *FAILED*.
- **createBattleInvitation(sender, receiver, battle)** Submits the information to create a new battle invitation.

  - *sender*: Sender student's email.
  - *receiver*: Invited student's email.
  - *tournament*: The specific battle of interest.

- **notifyBattleInvitation(sender, receiver, battle)** Starts the notification process regarding a particular battle invitation.
  - *sender*: Sender student's email.
  - *receiver*: Invited student's email.
  - *tournament*: The specific battle of interest.

- **getNotificationsPage(email)** Retrieves the notifications page related to a specific User.
  - *email*: Student's email.

- **addToGroupMembers(sender, receiver, battle)** Submits the information to insert the receiver of the battle invitation in the same group of the sender, within the context of a specific battle.
  - *sender*: Sender student's email.
  - *receiver*: Invited student's email.
  - *battle*: The specific battle of interest.

- **modifyBattleInvitation(sender, receiver, battle, status)** Submits the information to update the status of a specific battle invitation.
  - *sender*: Sender student's email.
  - *receiver*: Invited student's email.
  - *battle*: The specific battle of interest.
  - *status*: The invitation status, which can be either *ACCEPTED* or *REJECTED*.

- **notifyBattleInvitationResponse(sender, receiver, battle, status)** Starts the notification process regarding the acceptance or rejection of a specific battle invitation.
  - *sender*: Sender student's email.
  - *receiver*: Invited student's email.

  - *battle*: The specific battle of interest.
  - *status*: The invitation status, which can be either *ACCEPTED* or *REJECTED*.

- **getPersonalAreaPage(email)** Retrieves the personal area page related to a specific User.
  - *email*: User's email.
- **modifyStudentInfo(student, newInfo)** Submits the information to update the information related to a specific student.
  - *student*: Student's email.
  - *newInfo*: A set of information to change.
- **createBattle(creator, battleInfo)** Submits the information to create a new battle.
  - *creator*: Battle creator's email.
  - *battleInfo*: A set of input parameters referred to the battle attributes, as minimum and maximum number of participants per team, etc.
- **closeTournament(tournament)** Starts the closing process of a specific tournament.
  - *tournament*: The tournament to close.
- **getProjectPage(battle)** Retrieves the project page related to a specific battle.
  - *battle*: The specific battle of interest.
- **addManualScore(group, pointsToAdd, battle)** Submits the information to update a group's score by adding a specific number of points.
  - *group*: The group to which points will be added.
  - *pointsToAdd*: The quantity of points to be added.
  - *battle*: The relevant battle.
- **closeBattle(close)** Starts the losing process of a specific battle.
  - *battle*: The battle to close.

2.5.2. User Management Service Interfaces

- **checkIfAlreadyExists(email)** Queries the User Management Database to verify if the inserted email is already stored in the System.
  - *email*: Email to verify.
- **insertUser(registrationInfo)** Queries the User Management Database to create the new User.
  - *registrationInfo*: A set of input parameters referred to the submit form, as name, email, etc.
- **retrievePreferredLanguages(email)** Queries the User Management Database to retrieve all the student's preferred languages.
  - *email*: User's email.
- **insertBattleInvitation(sender, receiver, battle)** Queries the User Management Database to create a new battle invitation with status *PENDING*.
  - *sender*: Sender student's email.
  - *receiver*: Invited student's email.
  - *battle*: The specific battle of interest.
- **retrieveNotifications(email)** Queries the User Management Database to retrieve all the notifications related to a specific user.
  - *email* User's email.
- **retrievePersonalData(email)** Queries the User Management Database to retrieve User's personal data.
  - *email*: User's email.
- **updateStudentInfo(student, newInfo)** Queries the User Management Database to update the information related to a specific student.
  - *student*: Student's email.
  - *newInfo*: A set of information to change.

2.5.3. Email Service Provider Interfaces

- **sendConfirmationEmail(email)** Triggers the Email Provider to send the confirmation email to the new User.
  - *email*: Email address to which the confirmation email will be sent.
- **sendEmailToAllStudents(about)** Triggers the Email Provider to send an email to all the platform students.
  - *about*: Specifies the information to be sent within the email body.
- **sendEmailToTeamStudent(student, battle, state)** Triggers the Email Provider to send an email to a specific student about his enrollment state in a certain battle.
  - *student*: Student's email.
  - *battle*: The specific battle of interest.
  - *state*: The enrollment status, which can be either *SUCCESS* or *FAILED*.
- **sendBattleInvitationEmail(sender, receiver, battle)** Triggers the Email Provider to send an email invitation from the sender to the receiver for participation in the specified battle
  - *sender*: Sender student's email.
  - *receiver*: Invited student's email.
  - *battle*: The specific battle of interest.
- **sendEmailAboutBattleResponse(sender, receiver, battle, status)** Triggers the Email Provider to send an email regarding the acceptance or rejection of a specific battle invitation.
  - *sender*: Sender student's email.
  - *receiver*: Invited student's email.
  - *battle*: The specific battle of interest.
  - *status*: The invitation status, which can be either *ACCEPTED* or *REJECTED*.
- **sendScoreEmail(group, battle)** Triggers the Email Provider to send an email to a certain group, regarding the score update of a specified battle.
  - *group*: The group whose score has been calculated.
  - *battle*: The battle of interest.

- **sendClosedTournamentNotification(tournament)** Triggers the Email Provider to send an email to all the students who participated in the recently concluded tournament, notifying them of the tournament's closure.
  - *tournament*: The tournament that has recently concluded.

  sendClosedBattleNotification(battle) Triggers the Email Provider to send an email to all the students who participated in the recently concluded battle, notifying them of the battle's closure.
  - *battle*: The battle that has recently concluded.

2.5.4. Tournament Management Service Interfaces

- **insertTournament(creator, tournamentInfo)** Queries the Tournament Database to create a new Tournament.
  - *creator*: Tournament creator's email.
  - *tournamentInfo*: A set of input parameters referred to the tournament attributes, as register deadline, etc.
- **insertInvitation(sender, receiver, tournament)** Queries the Tournament Database to create a new tournament help invitation, saving it with status *PENDING*.
  - *sender*: Tournament's creator email.
  - *receiver*: Invited educator's email.
  - *tournament*: The specific tournament of interest.
- **retrieveMachingTournaments(lang)** Queries the Tournament Database to retrieve all the tournaments whose programming language matches the one entered as a parameter.
  - *lang*: Programming language used as the criterion for matching tournaments.
- **insertTournamentEnrollment(email, tournament)** Queries the Tournament Database to create a new tournament enrollment.
  - *email*: Enrolled student's email.
  - *tournament*: The specific tournament of interest.
- **retrieveTournamentDetails(tournament)** Queries the Tournament Database to retrieve all the details related to a specific tournament.

  - *tournament*: The specific tournament of interest.

- **retrieveAssociatedTournaments(email)** Queries to Tournament Database to retrieve all the tournaments associated to a specific User.
  - *email*: User's email.

- **updateTournamentStatus(tournament, status)** Queries to Tournament Database an update to the status of a particular tournament.
  - *tournament*: The specific tournament for which the status is to be updated.
  - *status*: The new status that the tournament will be assigned.

2.5.5. Notification Service Interfaces

- **sendNotificationToAllStudents(about)** Triggers the Notification Service to send a built-in notification to all the platform participants.
  - *about*: Specifies the information to be sent within the notification body.

- **sendNotificationToTeamStudent(student, battle, state)** Triggers the Notification Service to send a built-in notification to a specific student about his enrollment state in a certain battle.
  - *student*: Student's email.
  - *battle*: The specific battle of interest.
  - *state*: The enrollment status, which can be either *SUCCESS* or *FAILED*.

- **sendBattleInvitationNotification(sender, receiver, battle)** Triggers the Notification Service to send a built-in notification from the sender to the receiver for participation in the specified battle.
  - *sender*: Sender student's email.
  - *receiver*: Invited student's email.
  - *battle*: The specific battle of interest.

- **sendNotificationAboutBattleResponse(sender, receiver, battle, status)** Triggers the Notification Service to send a built-in notification regarding the acceptance or rejection of a specific battle invitation.
  - *sender*: Sender student's email.

  - *receiver*: Invited student's email.
  - *battle*: The specific battle of interest.
  - *status*: The invitation status, which can be either *ACCEPTED* or *REJECTED*.

- **sendScoreNotification(group, battle)** Triggers the Notification Service to send a built-in notification to a certain group, regarding the score update of the specified battle.
  - *group*: The group whose score has been calculated.
  - *battle*: The battle of interest.

- **sendClosedTournamentEmail(tournament)** Triggers the Notification Service to send a built-in notification to all the students who participated in the recently concluded tournament, notifying them of the tournament's closure.
  - *tournament*: The tournament that has recently concluded.

  sendClosedBattleEmail(battle) Triggers the Notification Service to send a built-in notification to all the students who participated in the recently concluded battle, notifying them of the battle's closure.
  - *battle*: The battle that has recently concluded.

2.5.6. Battle Management Service Interfaces

- **retrieveAssociatedBattles(tournament)** Queries the Battle Database to retrieve all the battles related to a specific tournament.
  - *tournament*: The specific tournament of interest.

- **retrieveBattleDetails(battle)** Queries the Battle Database to retrieve all the details related to a specific battle.
  - *battle*: The specific battle of interest.

- **insertSoloParticipation(student, battle)** Queries the Battle Database to create a new individual battle participation.
  - *student*: Student's email.
  - *battle*: The specific battle of interest.

- **updateGroupMembers(sender, receiver, battle)** Queries the Battle Database to update the members of a group, inserting the receiver as new member.

  - *sender*: Sender student's email.
  - *receiver*: Invited student's email.
  - *battle*: The specific battle of interest.

- **updateBattleInvitation(sender, receiver, battle, status)** Queries the Battle Database to update the status of a specific battle invitation.
  - *sender*: Sender student's email.
  - *receiver*: Invited student's email.
  - *battle*: The specific battle of interest.
  - *status*: The invitation status, which can be either *ACCEPTED* or *REJECTED*.

- **insertBattle(creator, battleInfo)** Queries the Battle Database to create a new battle.
  - *creator*: Battle creator's email.
  - *battleInfo*: A set of input parameters referred to the battle attributes, as minimum and maximum number of participants per team, etc.

- **retrieveGroupSources(group, battle)** Queries the Battle Database to retrieve the sources related to a certain group, involved in a certain battle.
  - *group*: The group of interest.
  - *battle*: The battle of interest.

- **updateBattleStatus(battle, status)** Queries to Battle Database an update to the status of a particular battle.
  - *battle*: The specific battle for which the status is to be updated.
  - *status*: The new status that the battle will be assigned.

2.5.7. Score Calculation Service Interfaces

- **computeScore(group, groupSources, battle)** Computes the score associated to a certain group.
  - *group*: The group of interest.
  - *groupSources*: The sources uploaded by the group with the commit.
  - *battle*: The battle of interest.

- **updateScore(group, pointsToAdd, battle)** Queries the Scoring Database to update a group's score by adding a specific number of points.
  - *group*: The group to which points will be added.
  - *pointsToAdd*: The quantity of points to be added.
  - *battle*: The relevant battle.

2.5.8. Static Analysis tools Interfaces

- **retrieveQualityScore(groupSources, aspect)** Computes the score associated to a certain quality aspect, analyzed on the specific sources.
  - *groupSources*: The sources uploaded by the group with the commit.
  - *aspect*: The aspect of interest. Can be either *SECURITY*, or *MAINTAINABILITY*, or *RELIABILITY*.

2.5.9. GitHub Service Interfaces

- **retrieveRelatedProjects(battle)** Retrieves all the uploaded sources referred to a specific battle, through GitHub.
  - *battle*: The battle of interest.

2.6. Selected Architectural Styles and Patterns

The CodeKataBattle system, as described in the previous sections, adopts several architectural styles and patterns to achieve its functionality and objectives. Below is an overview of these selected architectural styles and patterns:

- **Microservices Architecture**: This choice is driven by the need for scalability and flexibility in handling various independent functional modules such as user management, tournament organization, and battle management. Each microservice operates independently, ensuring that updates or changes in one service do not impact others, which is crucial for a system that requires frequent updates and enhancements.
- **Client-Server Architecture**: To facilitate user interaction with the platform. The web application serves as the client, offering an intuitive interface for students and educators to interact with the system. This separation of concerns allows for a focused development approach where the client side primarily deals with the presentation and user experience, while the server side (comprising various back-end services) manages the business logic and data handling.
- **Containerization**, using Docker, plays a pivotal role in the deployment of these microservices. By containerizing each service, the system ensures consistency in runtime environments and simplifies deployment processes.
- **orchestration tool** is chosen to manage containers efficiently. Kubernetes not only automates the deployment and scaling of these containers but also aids in maintaining the desired state of the system, a critical feature for an educational platform that might experience varying loads and usage patterns.
- **API Gateway Pattern**, streamlines the process of handling incoming requests. This gateway serves as the single point of entry for all requests coming into the system, effectively managing and routing these requests to the appropriate microservices. It simplifies the client's interaction with the backend services and also takes on additional responsibilities like security, rate-limiting, and request transformation, thus offloading these concerns from the individual microservices.
- **Load Balancing Pattern** is used to ensuce the high availability and responsiveness of the system. The load balancer efficiently distributes incoming traffic and requests across multiple servers, preventing any single server from becoming a bottleneck and ensuring a smooth user experience.

- **Circuit Braker**, is adopted to enhance the system's resilience and prevent failures from cascading through the services, the circuit breaker pattern is integrated within services that make external calls.
- **Database per service pattern**, almost all the microservices are paired with their own DB. This approach not only reduces the risk of data conflicts and enhances service independence but also allows for the use of different database technologies tailored to the specific needs of each service.

2.6.1. Other Design Decisions

- **Wide-Column Store Database** Offers several compelling advantages, particularly in terms of scalability, flexibility, and performance.
  - Firstly, wide-column stores efficiently handle large volumes of data and high throughput. They offer superior scalability compared to traditional relational databases, essential for accommodating the growing data needs of the platform.
  - Secondly, wide-column databases provide flexible schema design, which is particularly beneficial for the dynamic structure of the CKB platform.
  - Moreover, these databases excel in query performance, especially for read-heavy operations, which is crucial for the CKB platform's frequent data retrieval needs. They are optimized for high-speed read and write operations.
  - Additionally, wide-column stores support efficient data compression and storage, reducing storage costs while maintaining high data availability and integrity.
  - Lastly, wide-column databases align well with the microservices architecture of the CKB system. They offer robust integration capabilities, making it easier to interface with various microservices. This integration is key to maintaining the distributed, service-oriented structure of the platform, ensuring seamless interaction between different components and external services like GitHub and static code analysis tools.
- **Password Storing and User Authentication**, the login to CKBplatform requires a personal identifier and a password in order to guarantee confidentiality. For Educators, the ID coincides with the personal email address entered during registration. For Student, it corresponds either to the personal e-mail or the username they provided during the registration. For each registered entity, the access

passwords are hashed using bcrypt algorithm. Along with hashing, a unique salt (a random string) is added to each password before it's hashed. Then, the hashed and salted passwords are stored in the Database. No passwords are stored in clear text in the Database.

3 User Interface Design

3.1. Guest

3.1.1. Home Page

This interface appears when a Guest accesses the web application on his personal computer. The Guest can only navigate through this page, where are stored all the most relevant tournaments of the platform.

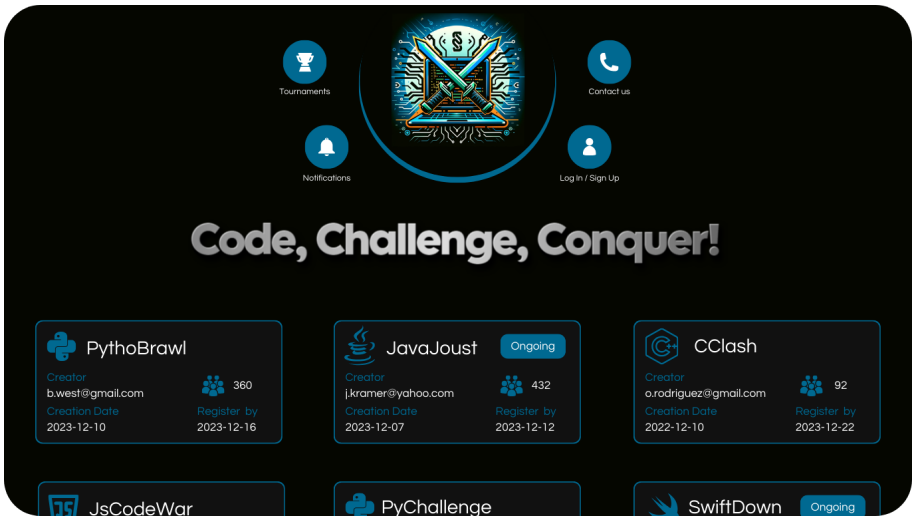

Figure 3.1: Guest - Home Page

3.1.2. Sign up / Log in Interface

This interface is displayed when a Guest clicks on *Sign Up/Log In* icon in the navigation bar. In this case, the Guest can choose either to Log In or Sign Up (figure 3.2). After making his decision, the Guest must choose whether to identify himself as an Educator or as a Student (figure 3.3).

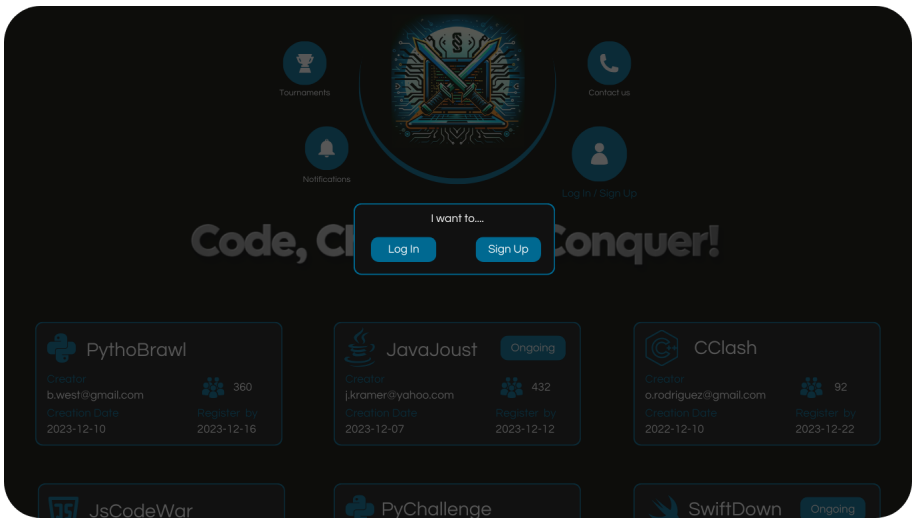

Figure 3.2: Guest - Sign up / Log in Interface 1

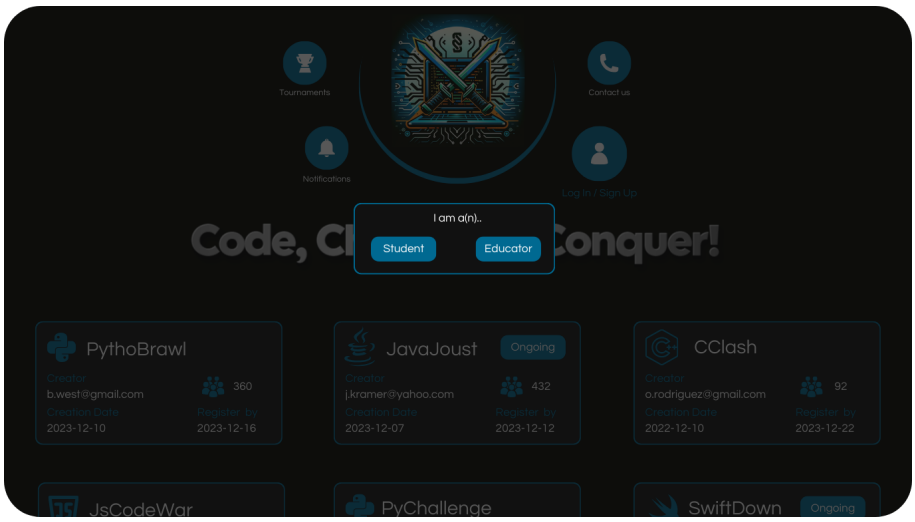

Figure 3.3: Guest - Sign up / Log in Interface 2

3.2. Educator

3.2.1. Sign Up Page

After clicking on the *Sign Up* and *Educator* options on the popups (respectively, figure 3.2 and 3.3), this page is displayed. It allows an Educator to register in the System.

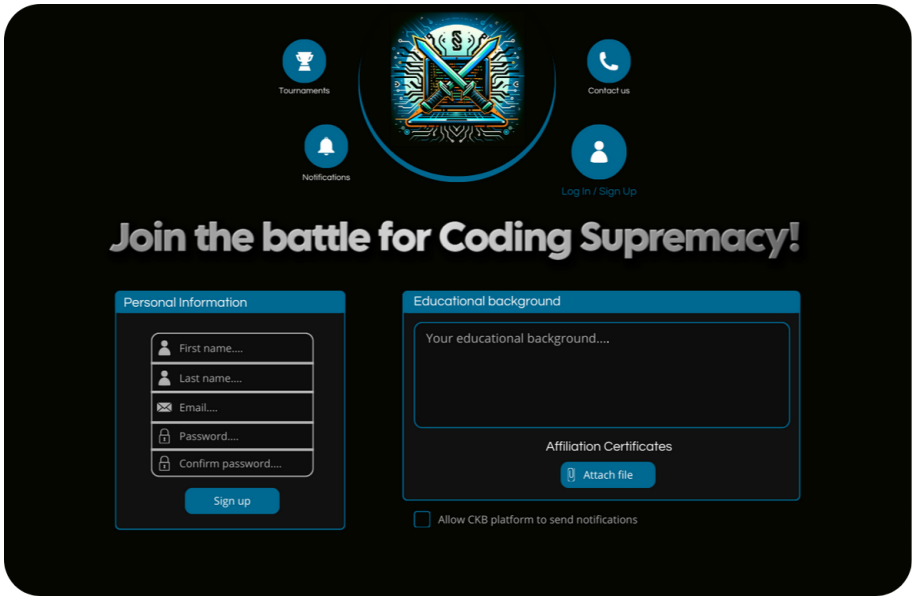

Figure 3.4: Educator - Sign Up Page

3.2.2. Log In Page

After clicking on the *Log In* and *Educator* options on the popups (respectively, figure 3.2 and 3.3), this page is displayed. It allows an Educator to log into the System.

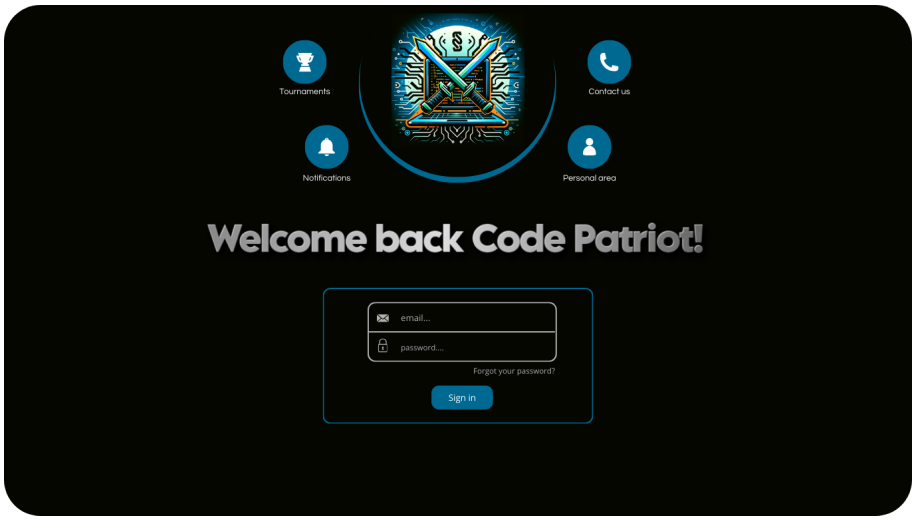

Figure 3.5: Educator - Log In Page

3.2.3. Home Page

This page appears when a Educator (already logged in) accesses the web application on his PC. Here he has the possibility to create new tournaments, through the *Create a new tournament* button.

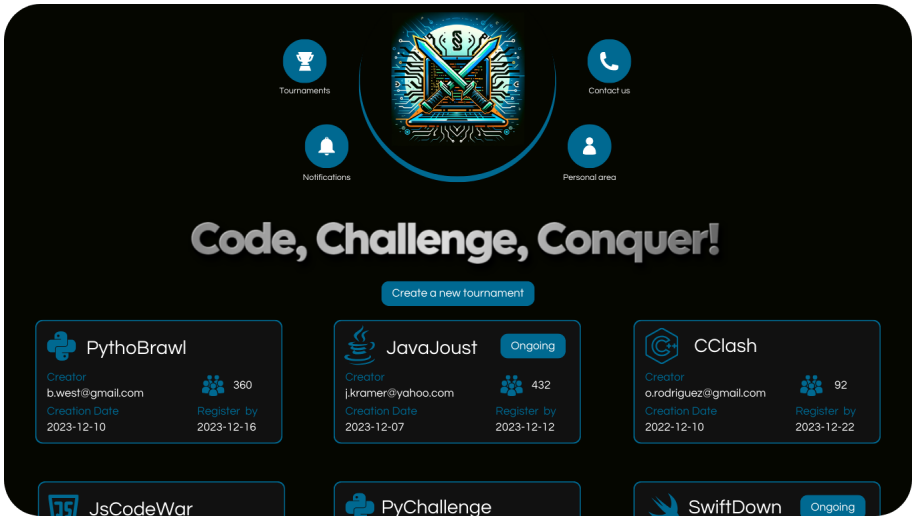

Figure 3.6: Educator - Home Page

3.2.4. Create a New Tournament Interface

This interface is displayed when a Educator clicks on *Create a New Tournament* button on the home page. The Educator must insert name, registration deadline, and programming language of the tournament he wants to create. He can eventually invite some educators

to help him, creating battles within the tournament (figure 3.17). After the Educator has entered the information and clicked on the *Create Tournament* button, a confirm popup interface is shown where the choices are summarized (figure 3.20). If the Educator confirms, the tournament is successfully created (figure 3.10).

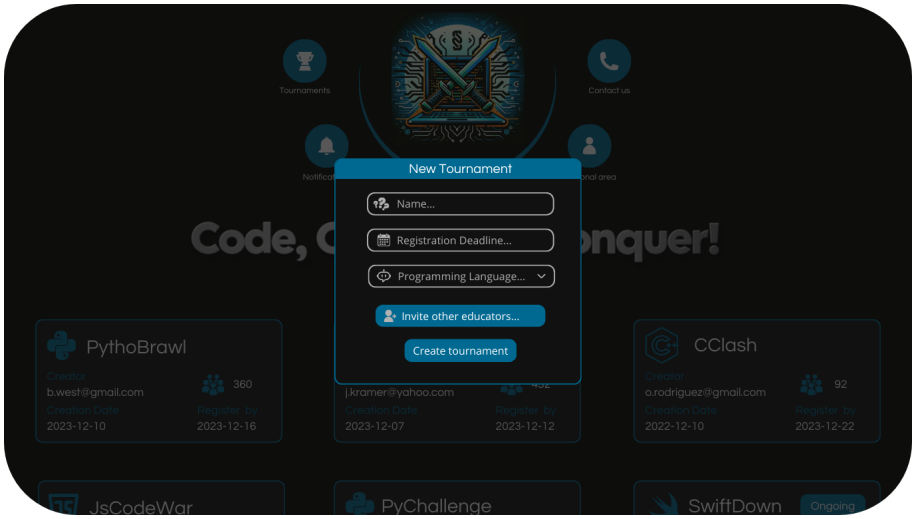

Figure 3.7: Educator - Creating a New Tournament Interface

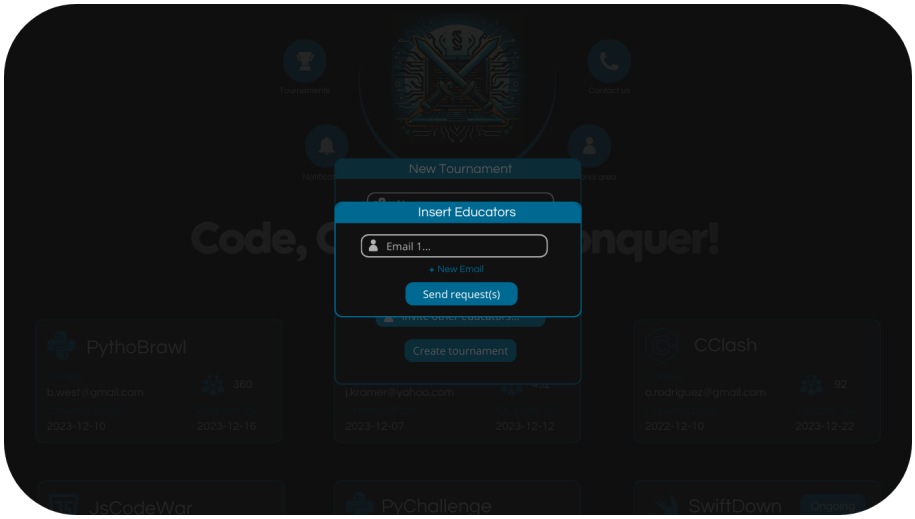

Figure 3.8: Educator - Insert Educators Interface

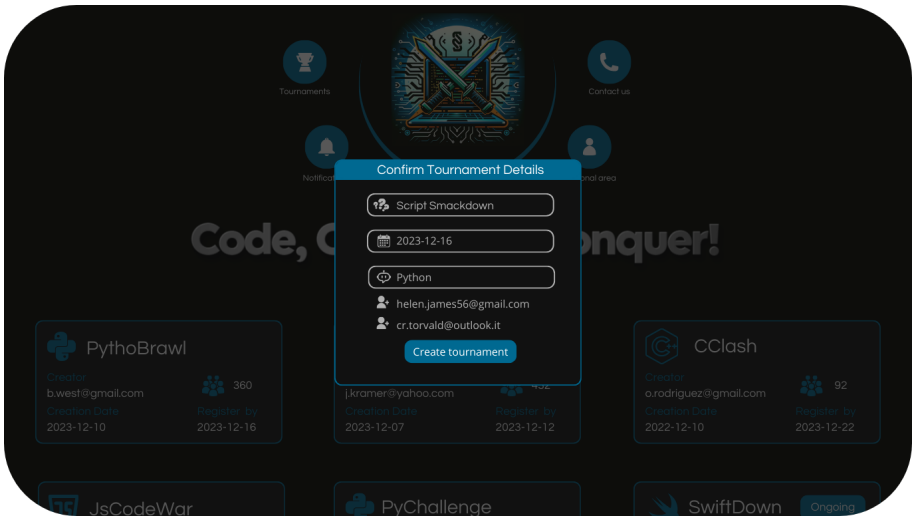

Figure 3.9: Educator - Confirm Tournament Creation Interface

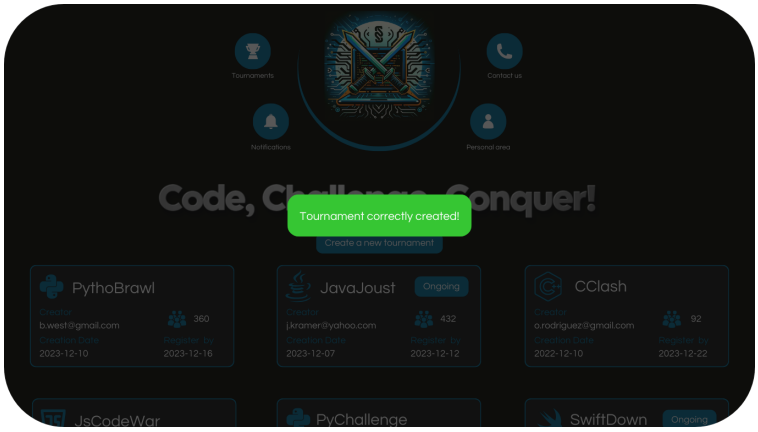

Figure 3.10: Educator - Tournament successfully created

3.2.5. Personal Area Page

This page is loaded when an Educator clicks on *Personal Area* icon in the navigation bar. It contains all the information about the Educator, including the tournaments he is enrolled in.

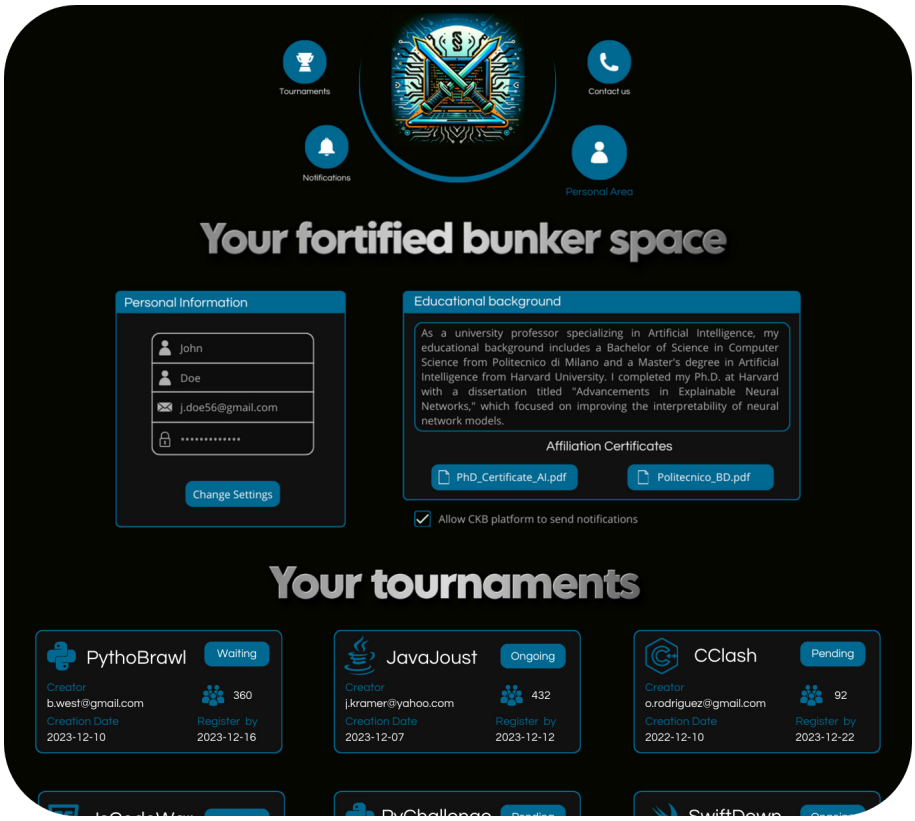

Figure 3.11: Educator - Personal Area Page

3.2.6. Change Settings Interface

This interface appears when an Educator clicks *Change Settings* button in his Personal Area Page. It allows him to change his personal information.

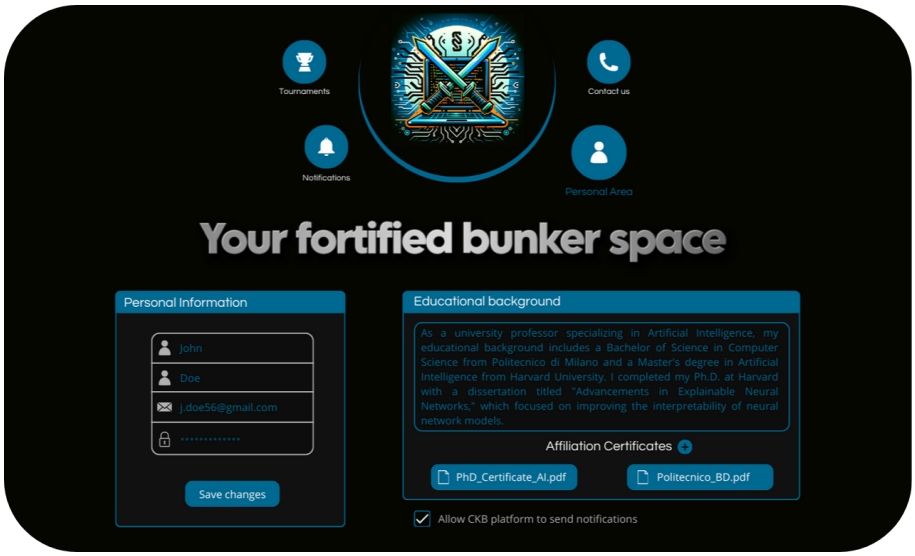

Figure 3.12: Educator - Change Settings Interface

3.2.7. Notifications Page

This page is loaded when an Educator clicks on *Notifications* icon in the navigation bar. Here he can find three kind of notifications:

- **New Challenges**: All the information about just created tournaments on the Platform.
- **Your challenges**: All the information about the tournaments (or battles) the Educator is enrolled in.
- **Invitations**: All the information about the invitations the Educator has received.

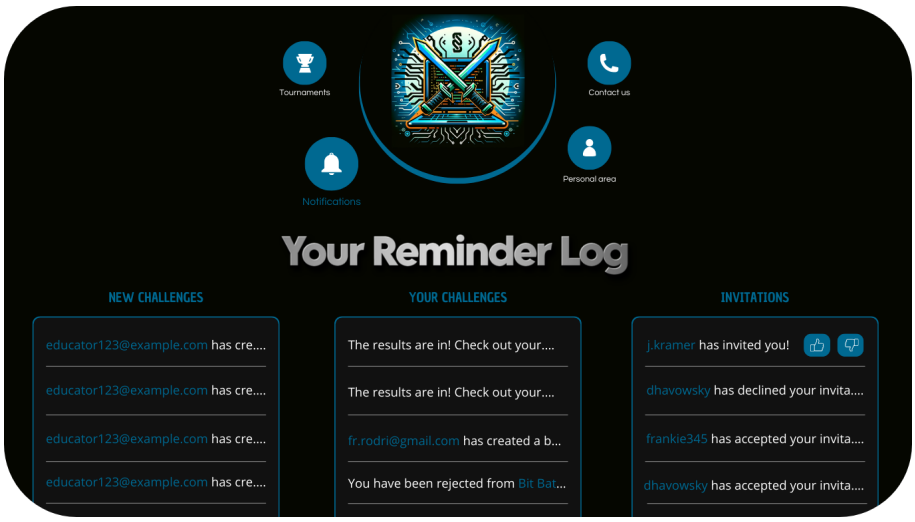

Figure 3.13: Educator - Notifications Page

3.2.8. Contact Us Page

This page is loaded when an Educator clicks on *Contact Us* icon in the navigation bar. Here he can find all the useful contacts of the Platform.

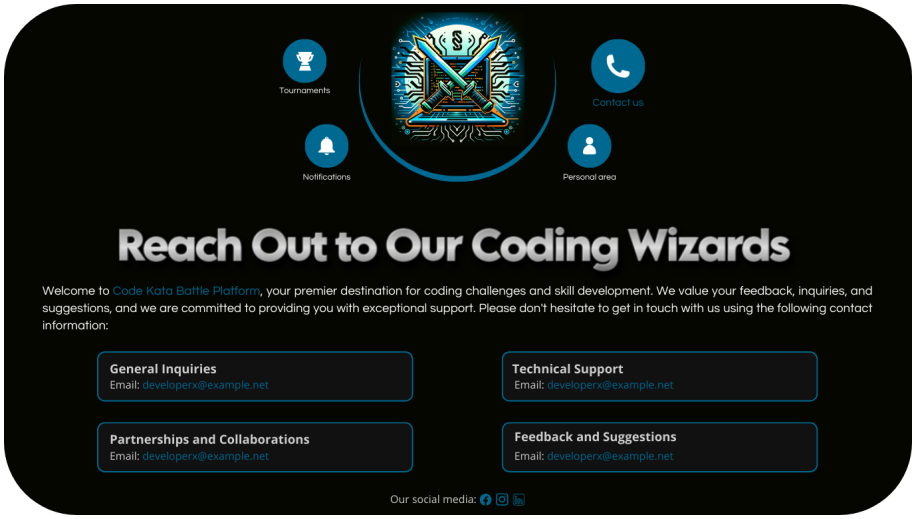

Figure 3.14: Educator - Contact Us Page

3.2.9. Available Tournaments Page

This page is loaded when an Educator clicks on *Tournaments* icon in the navigation bar. Here he can find all the tournaments created within the Platform, eventually through the research bar. The Educator can also navigate to the specific Tournament Page, by clicking on the relative card.

Figure 3.15: Educator - Available Tournaments Page

3.2.10. Tournament Page

This page is loaded when an Educator clicks on a specific tournament card. Here are stored all the useful information about the tournament, as its battles or its leader board. If the Educator has permissions, he can also create new battles, by clicking the *Create a New Battle*, or search one of them through the search bar. The *Load all...* button simply extends the leader board over the first ten positions.

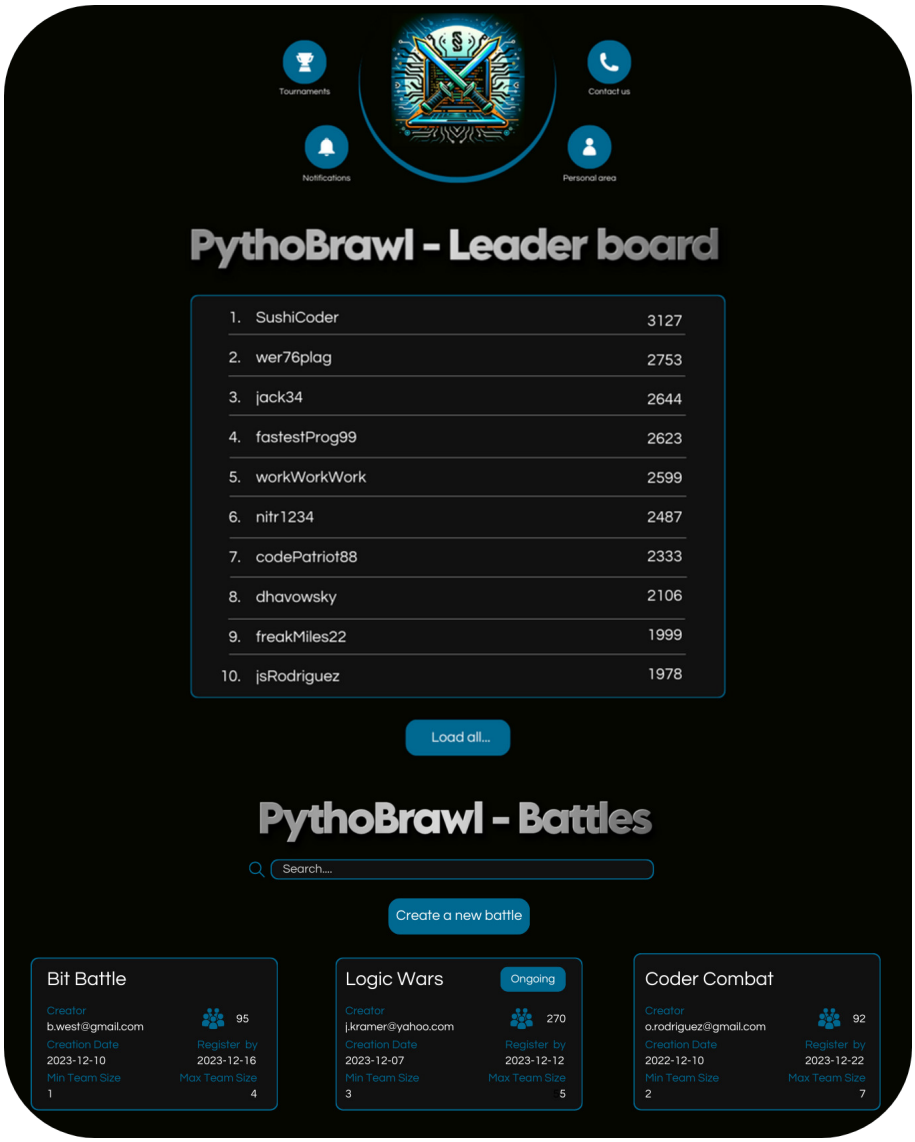

Figure 3.16: Educator - Tournament Page

3.2.11. Create a New Battle Interface

This interface is displayed when a Educator clicks on *Create a New Tournament* button on the home page. The Educator must insert all the relevant information about the battle he

wants to create. Once he has entered the information and clicked on the *Confirm* button, if all worked correctly, the battle is created (figure 3.10).

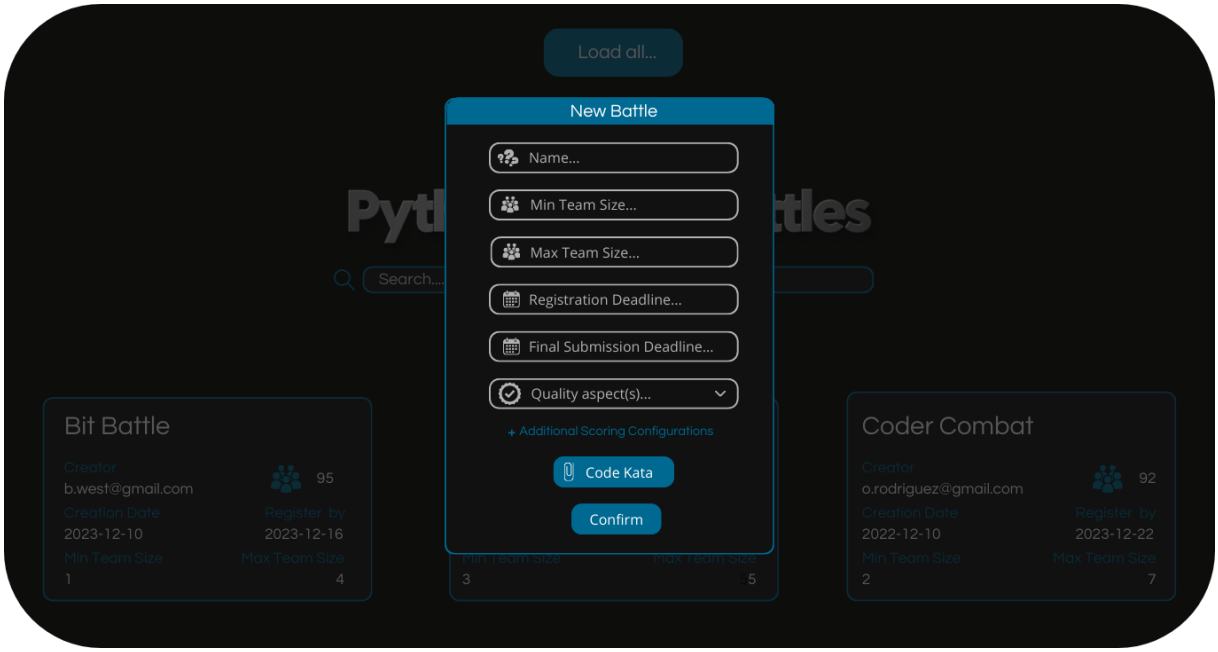

Figure 3.17: Educator - Create a New Battle Interface

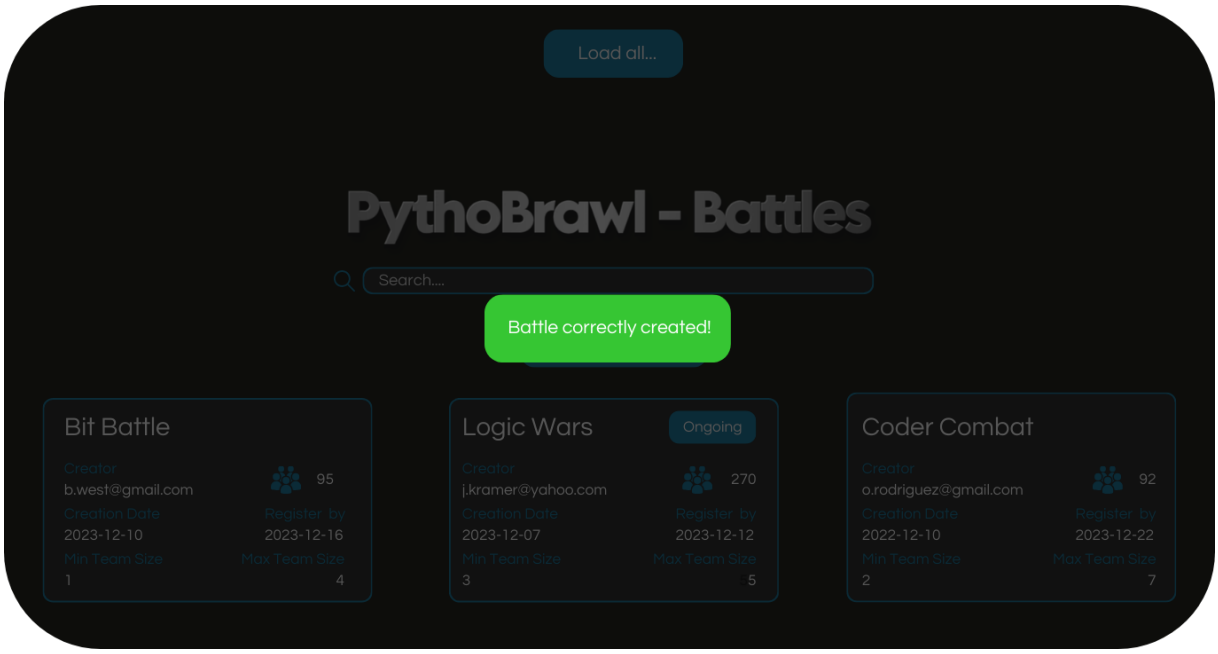

Figure 3.18: Educator - Battle successfully created

3.2.12. Battle Page

This page is loaded when an Educator clicks on a specific battle card within a tournament page. Here are stored all the useful information about the battle, as its details or its leader board. The *Load all...* button simply extends the leader board over the first ten positions.

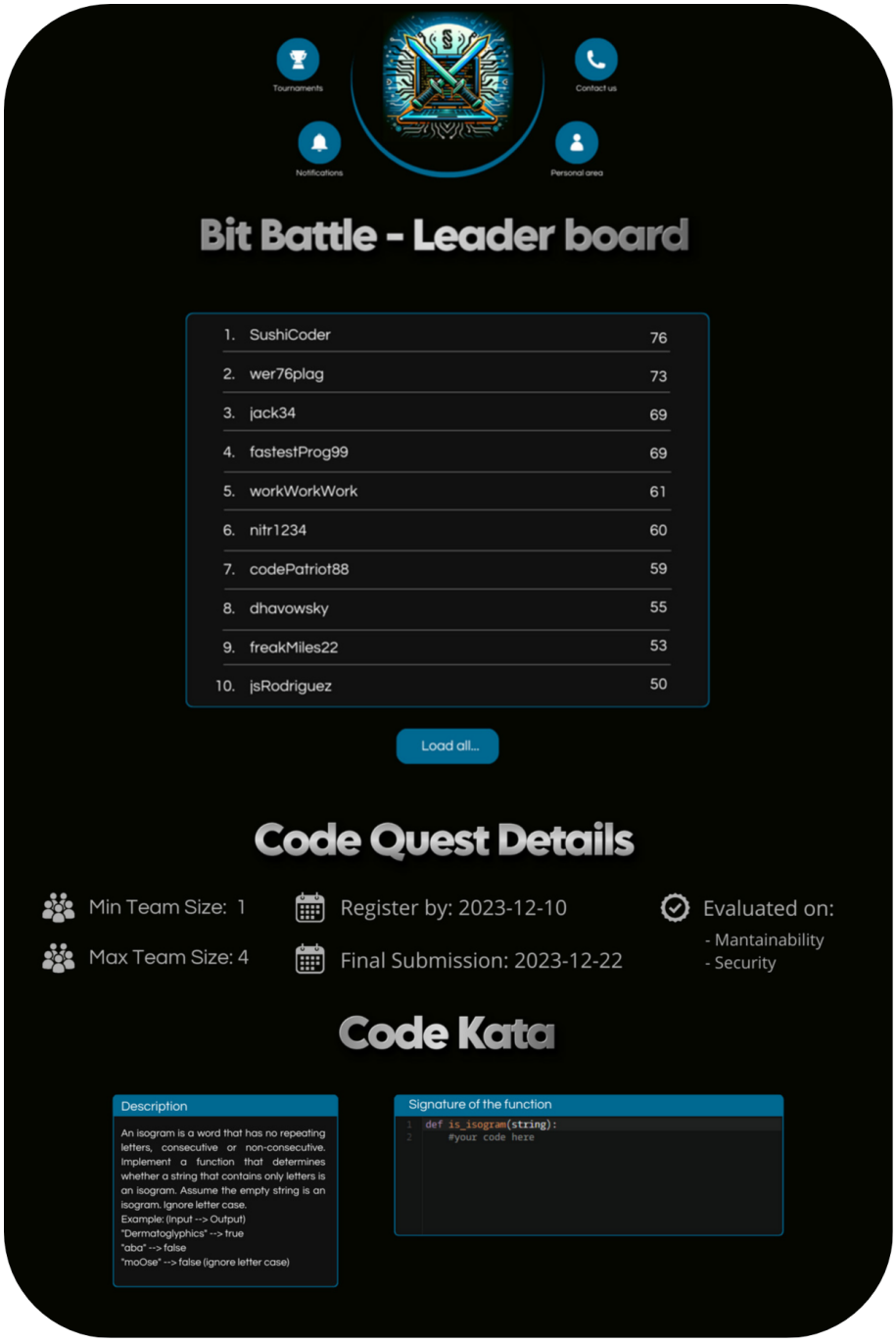

Figure 3.19: Educator - Battle Page

3.2.13. Project Page

This page is loaded when an Educator decides to evaluate manually the projects of a battle, whose final submission deadline has just expired. Here he can download each team's sources and make his evaluations, assigning his personal score.

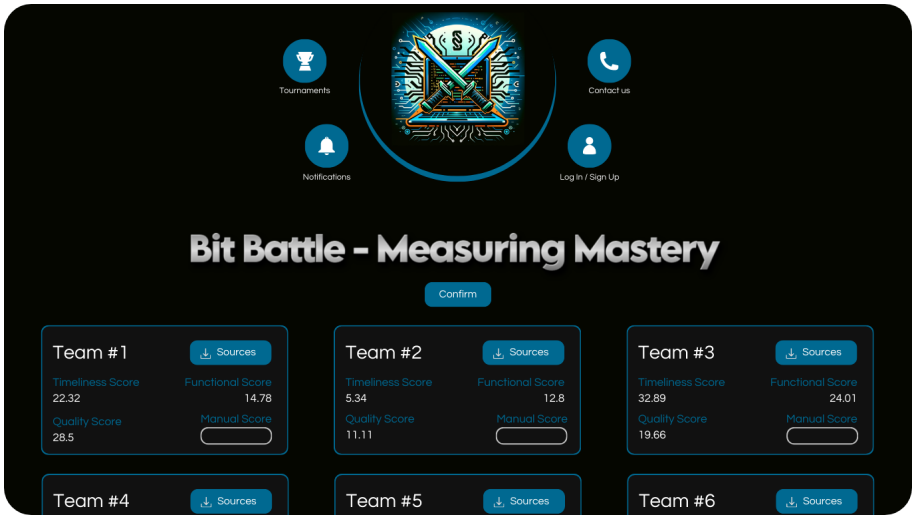

Figure 3.20: Educator - Project Page

3.3. Student

3.3.1. Sign Up Page

After clicking on the *Sign Up* and *Student* options on the popups (respectively, figure 3.2 and 3.3), this page is displayed. It allows a Student to register in the System.

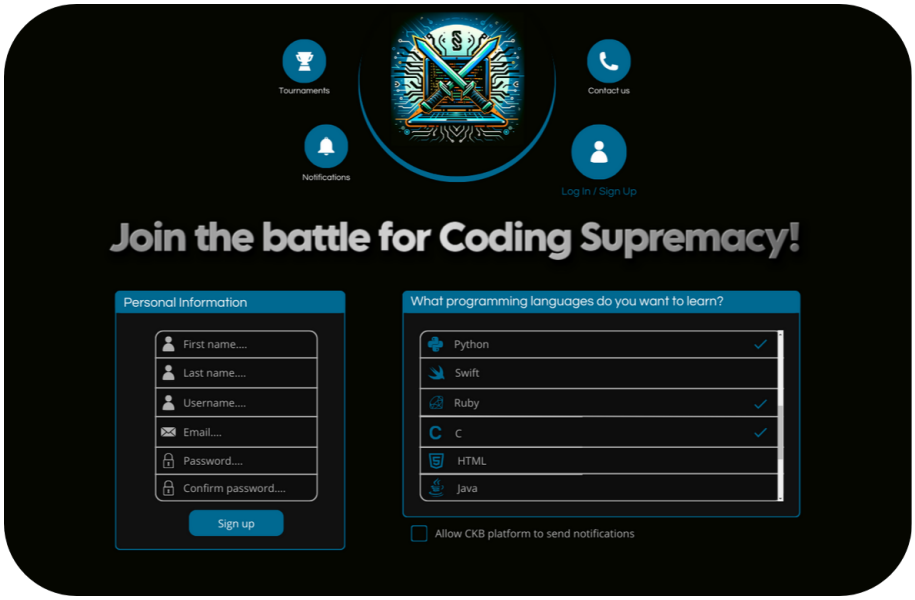

Figure 3.21: Student - Sign Up Page

3.3.2. Log In Page

After clicking on the *Log In* and *Student* options on the popups (respectively, figure 3.2 and 3.3), this page is displayed. It allows a Student to log into the System.

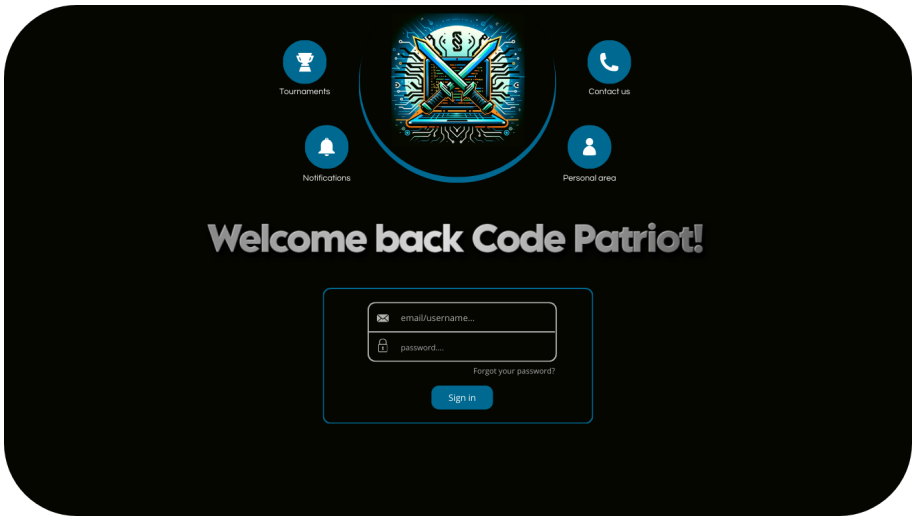

Figure 3.22: Student - Log In Page

3.3.3. Home Page

This page appears when a Student (already logged in) accesses the web application on his PC.

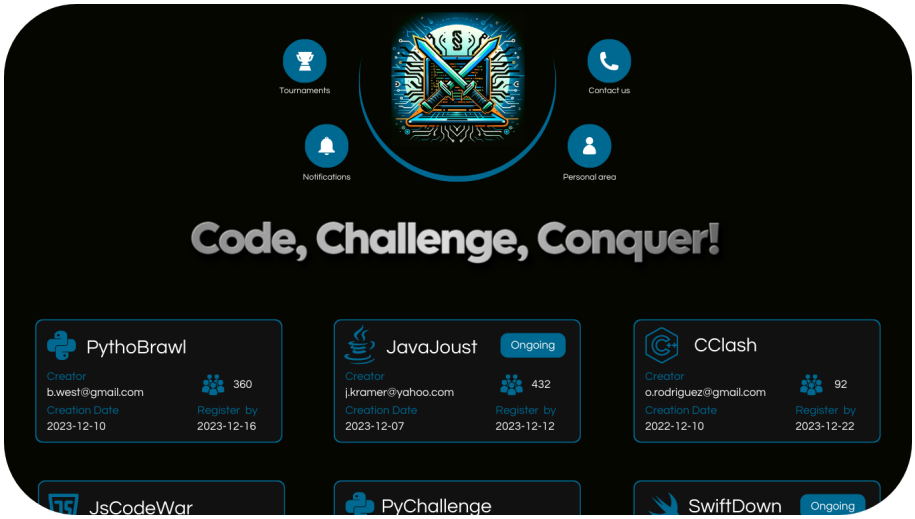

Figure 3.23: Student - Home Page

3.3.4. Personal Area Page

This page is loaded when a Student clicks on *Personal Area* icon in the navigation bar. It contains all the information about the Student, including the tournaments he is enrolled in.

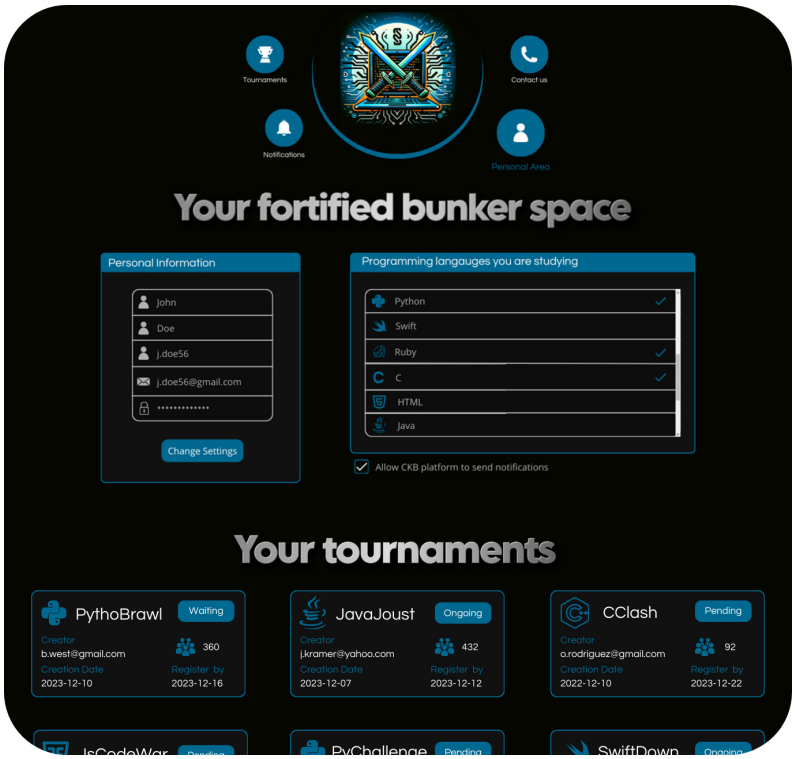

Figure 3.24: Student - Personal Area Page

3.3.5. Change Settings Interface

This interface appears when a Student clicks *Change Settings* button in his Personal Area Page. It allows him to change his personal information.

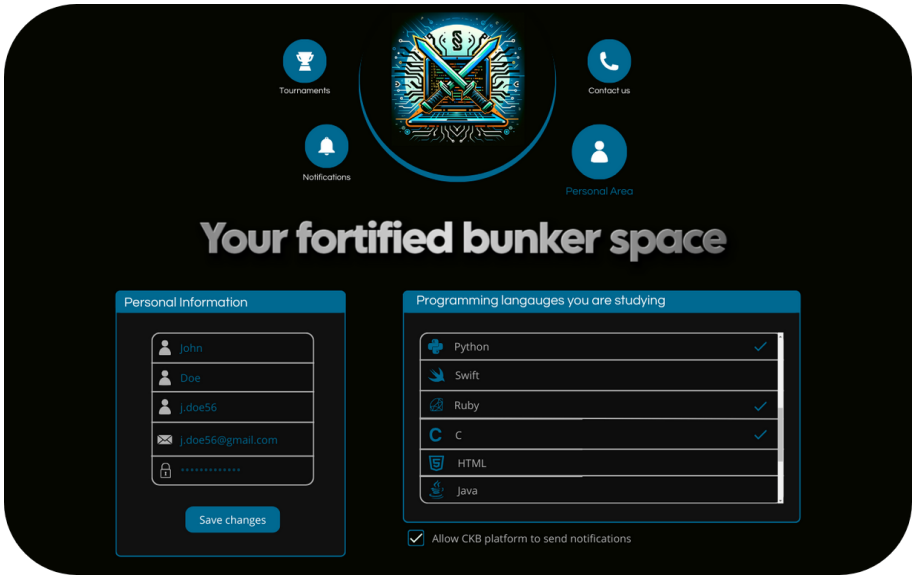

Figure 3.25: Student - Change Settings Interface

3.3.6. Notifications Page

This page is loaded when a Student clicks on *Notifications* icon in the navigation bar. Here he can find three kind of notifications:

- **New Challenges**: All the information about just created tournaments on the Platform.
- **Your challenges**: All the information about the tournaments (or battles) the Student is enrolled in.
- **Invitations**: All the information about the invitations the Student has received.

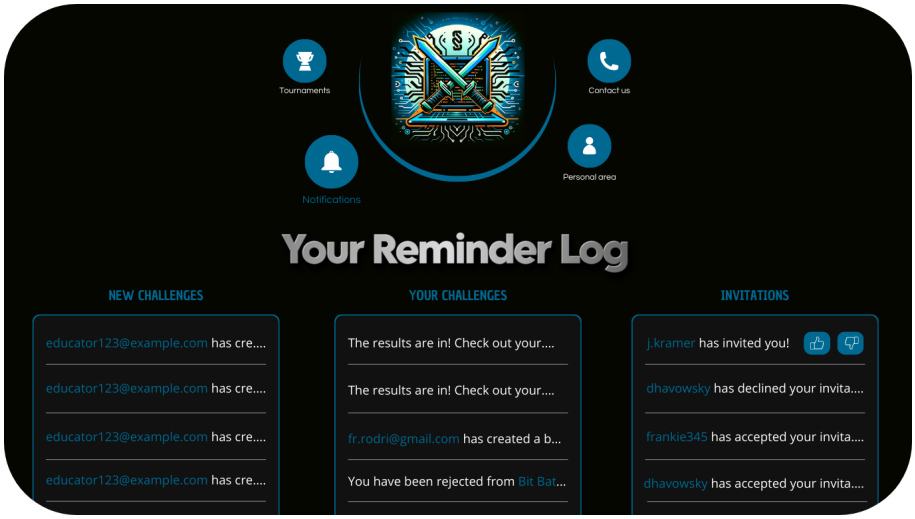

Figure 3.26: Student - Notifications Page

3.3.7. Contact Us Page

This page is loaded when a Student clicks on *Contact Us* icon in the navigation bar. Here he can find all the useful contacts of the Platform.

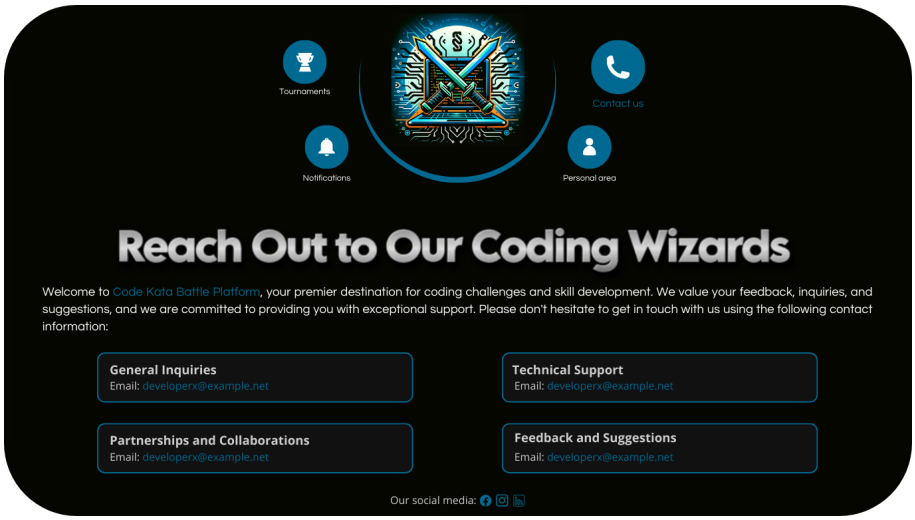

Figure 3.27: Educator - Contact Us Page

3.3.8. Available Tournaments Page

This page is loaded when a Student clicks on *Tournaments* icon in the navigation bar. Here he can find all the tournaments created within the Platform, eventually through the research bar. The Student can also navigate to the specific Tournament Page, by clicking on the relative card.

Figure 3.28: Student - Available Tournaments Page

3.3.9. Tournament Page

This page is loaded when a Student clicks on a specific tournament card. Here are stored all the useful information about the tournament, as its battles or its leader board. The

student can search one specific battle through the search bar. The *Load all...* button simply extends the leader board over the first ten positions.

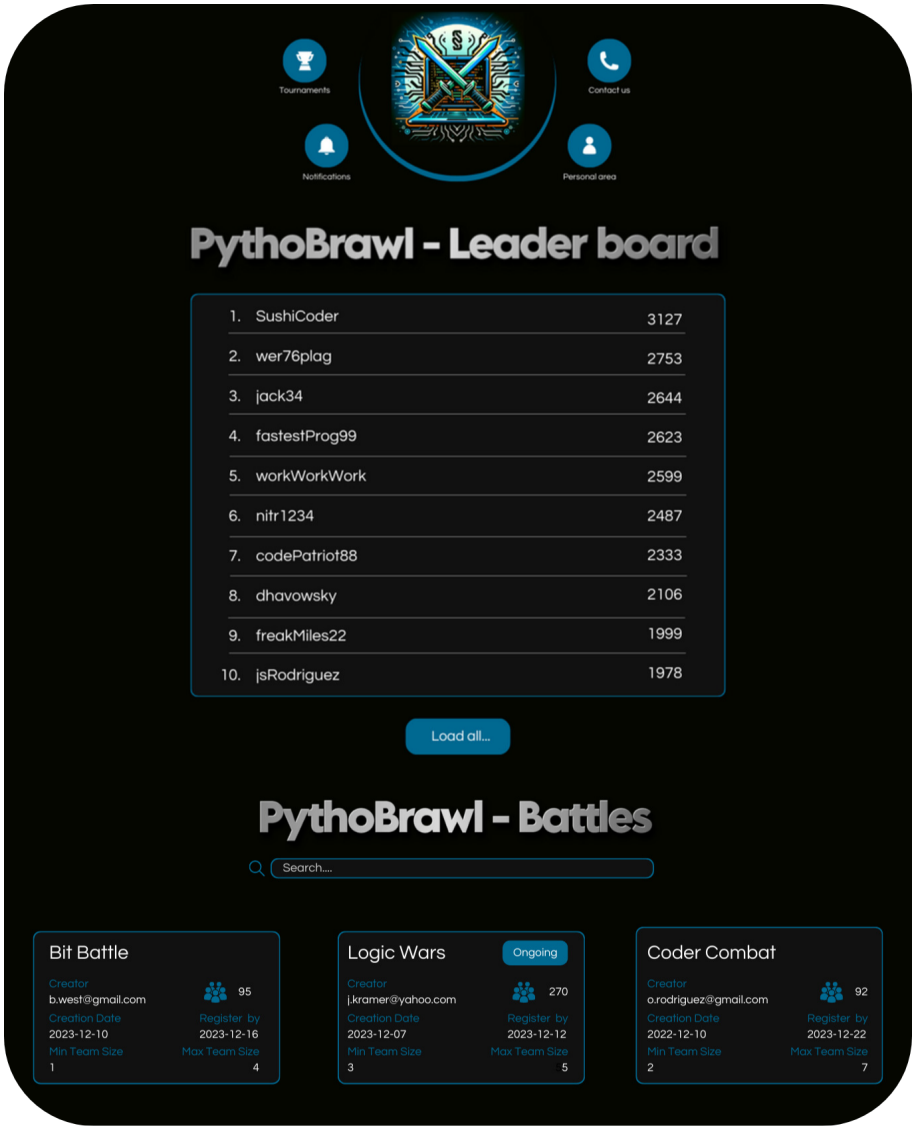

Figure 3.29: Student - Tournament Page

3.3.10. Battle Page

This page is loaded when a Student clicks on a specific battle card within a tournament page. Here are stored all the useful information about the battle, as its details or its leader board. The *Load all...* button simply extends the leader board over the first ten positions.

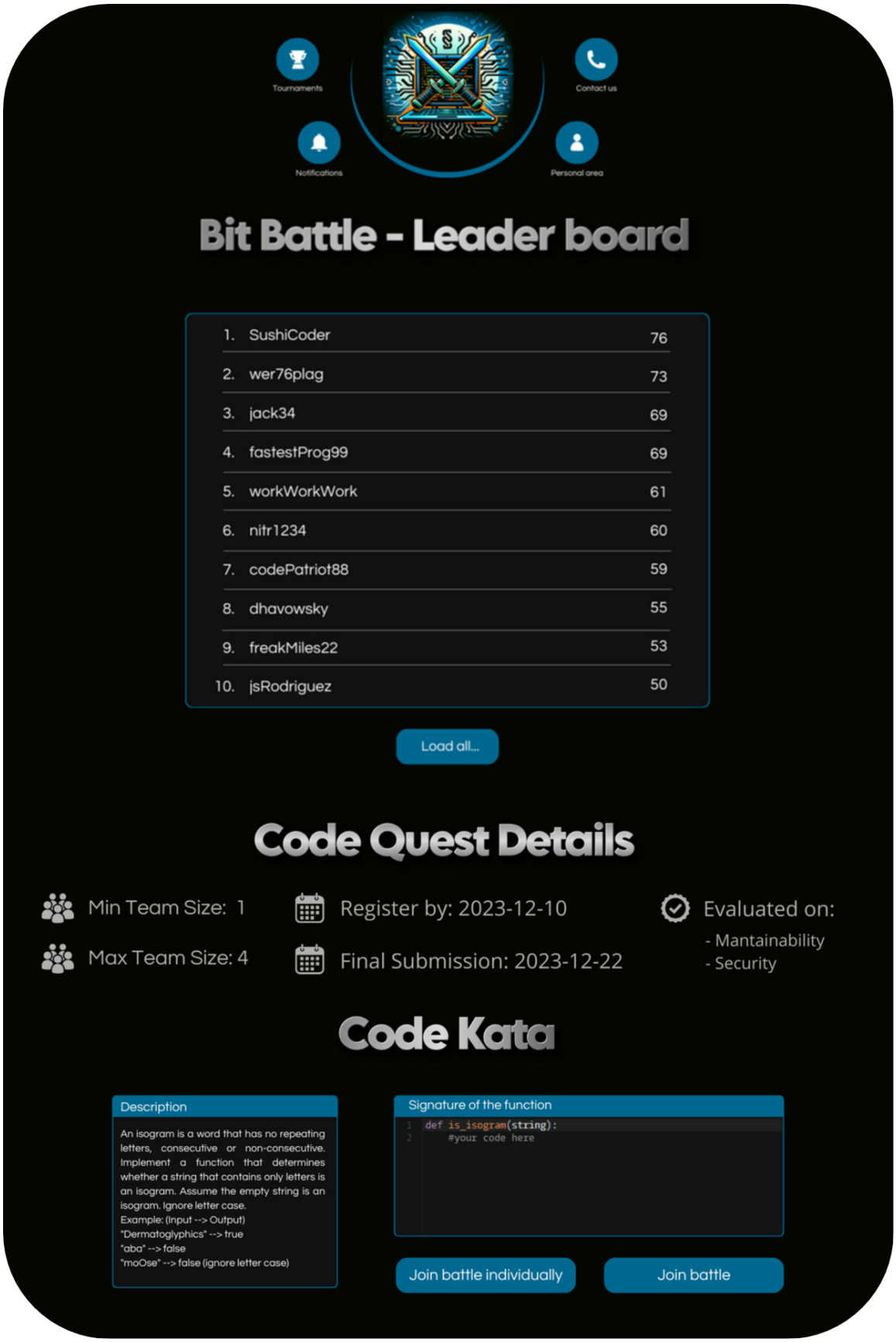

Figure 3.30: Student - Battle Page

3.3.11. Join Battle Interface

A Student can join a battle through the relative page:

- **Individually**: this option is available if and only if the battle has min team size equals to one and is chosen by clicking the *Join battle individually* button.
- **Normally**: this option is chosen by clicking the *Join battle* button (figure 3.31). In this case, he must also choose between:
  - **Available to join a group**: The System will insert the Student in a random team.

  Invite other students: The system will attempt to form a team with the invited students if the number of accepting students falls within the minimum and maximum group size limits. The system will attempt to form a team with the invited students if the number of accepting students falls within the minimum and maximum group size limits. In this case, the student must insert at least one student's email that he wants to invite (figure 3.32).

If all worked correctly, the battle request is created (figure 3.33).

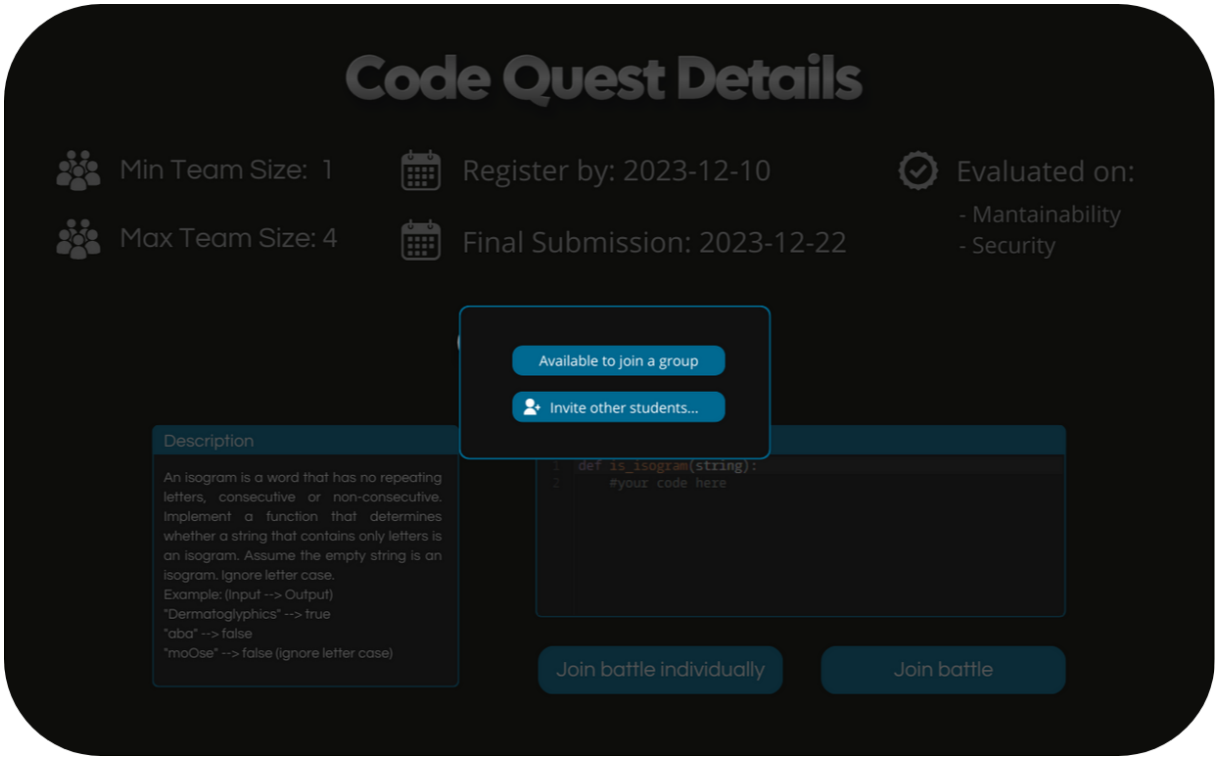

Figure 3.31: Student - Join Battle Interface

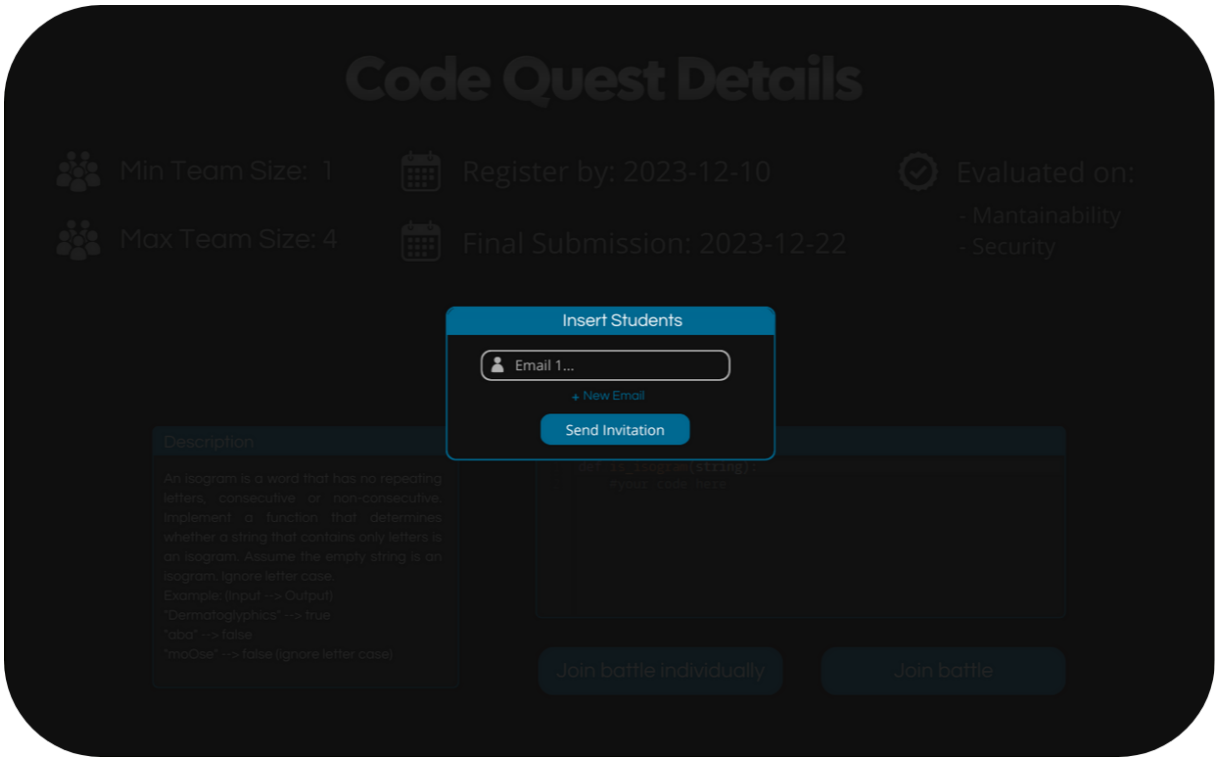

Figure 3.32: Student - Insert Students Interface

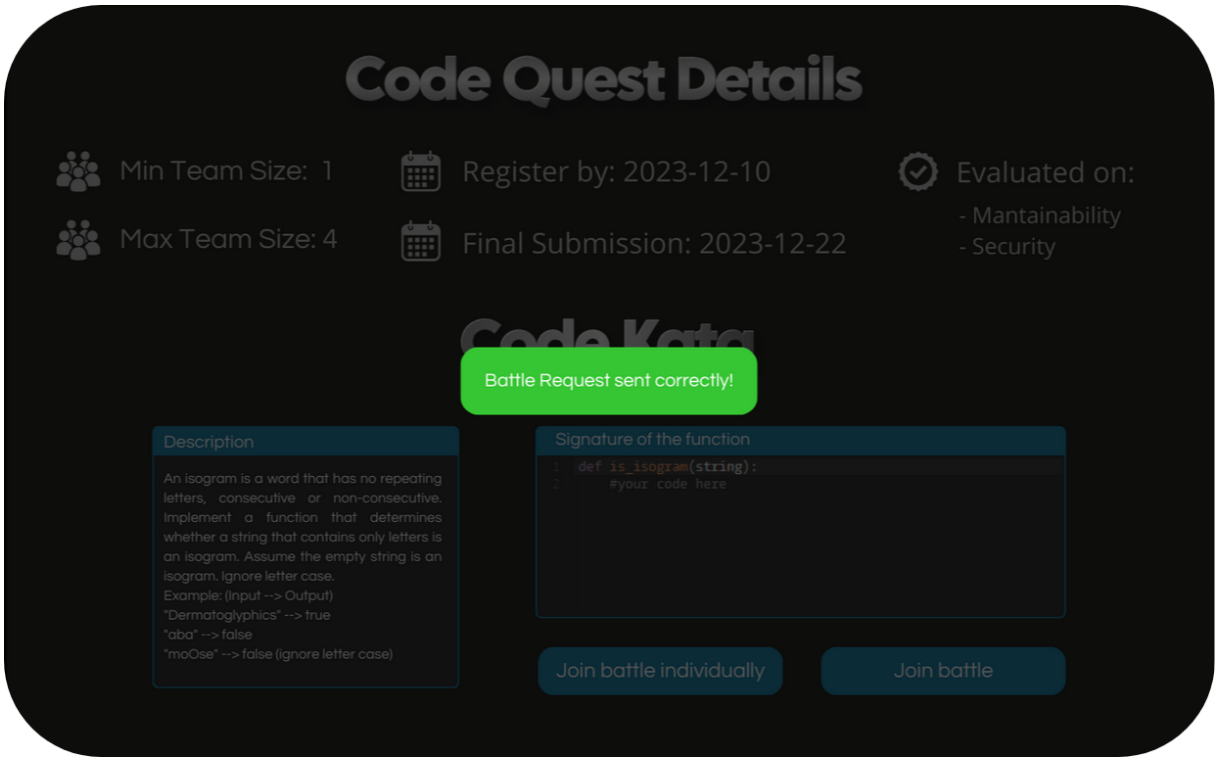

Figure 3.33: Student - Battle request correctly sent

4 | Requirements traceability

In this paragraph is shown the mapping between every component and the requirement satisfied.

	User Account Creation and Management
Requirements	• **RE1**: The System allows users to create either a Student or an Educator • **RE2**: An Educator account can be created if and only if he/she provides her educational background and institution affiliation • **RE3**: The System must allow a User to register by filling in a form containing a set of fields • **RE3.1**: A User must be able to indicate his personal data: his first name, last name, an e-mail, and a password • **RE4**: The System must allow a Student to enter a username • **RE5**: The System must allow a Guest, when registering as an educator, to enter his educational background and institutional affiliation • **RE7**: The System must store the registration data relating to the registration • **RE8**: The System must allow a user to login by entering his email and password • **RE9**: The System must allow a student to login by entering his email/username and password • **RE10**: The System must ask a Guest permission to send notifications • **RE11**: The System must process the Guest's choice about the notification permission • **RE59**: The System must guarantee secure login functionality for both students and educators
Components	User Management Service, including sub-components for User Registration, Profile Management, and Authentication.

Table 4.1: User Account Creation and Management

	Tournament and Battle Management - 1
Requirements	• **RE6**: The System must grant the possibility to create a tournament only to Educators • **RE12**: The System must allow a Participant to search the battles by name • **RE13**: The System must allow a Participant to search the tournaments • **RE15**: The System must allow the Educator that sets up a tournament to grant privileges to other educators to create battles within it • **RE17**: The System must allow Students to visualize the battles in a tournament if and only if they have subscribed to it • **RE18**:The System must allow to all subscribed Users to visualize all the ongoing tournaments and their related rank • **RE19**: The System must allow Students to enroll in existing tournament, if and only if the expiration date is not met yet • **RE20**: The System must allow Educators to create tournaments on the CKB platform • **RE21**: The System must allow Educators to define and publish code katas within a tournament • **RE22**: The System must allow an Educators to define and publish code katas within a tournament by filling in a form containing a set of fields • **RE22.1**: An Educator must be able to indicate in a code kata definition: programming language, project details, test cases, and automation scripts • **RE22.2**: An Educator must be able to set minimum and maximum numbers of students per group for each battle • **RE22.3**: An Educator must be able to set minimum and maximum numbers of students per group for each battle • **RE22.4**: An Educator must be able to set additional scoring configuration • **RE22.5**: An Educator must be able to select aspects to take into consideration to evaluate quality level of the source
Components	Tournament Management Service and Battle Management Service, including sub-components for creation, management, subscription, and scoring of tournaments and battles.

Table 4.2: Tournament and Battle Management - 1

	Tournament and Battle Management - 2
Requirements	• **RE23**: The System must allow Students to invite other Students to join a battle with them, if and only if the invited Students already have an account • **RE24**: The System must not allow Students to invite more friends in a battle than the maximum number required to build a team • **RE25**: The System must allow invited students to either accept or reject the invitation of joining a battle • **RE26**: The System must not allow a Student to join a closed battle • **RE27**: The System must not allow an Educator without "creator" privileges in a tournament to create a battle within it • **RE28**: The System must not allow a Student to join a battle, if the expiration date has come • **RE52**: The System must set to "Created" the status of a tournament just created by an Educator. • **RE53**: The System must change to "Opened" the status of a tournament when its registration deadline expires. • **RE54**: The System must change to "Closed" the status of a tournament when its creator closes it. • **RE55**: The System must set to "Created" the status of a battle just created by an Educator. • **RE56**: The System must change to "Opened" the status of a battle when its registration deadline expires. • **RE57**: The System must change to "Evaluating" the status of a battle when its final submission deadline expires. • **RE58**: The System must change to "Closed" the status of a battle once battle scores have been calculated.
Components	Tournament Management Service and Battle Management Service, including sub-components for creation, management, subscription, and scoring of tournaments and battles (Score Calculation Service).

Table 4.3: Tournament and Battle Management - 2

	Score Calculation and Evaluation
Requirements	• **RE32**: The System must be able to update battle score, after each student's commit • **RE33**: The System must allow Educators to perform manual evaluation to the scoring of teams, after each battle has ended • **RE34**: The System must allow Educators to have access to all the source codes of each team in order to perform the manual evaluation • **RE35**: The System must guarantee to students and educators, a continuous update of battle scores and ranks • **RE42**: The System must compute the automated evaluation after each push based on mandatory factors: functional aspects, timeliness, quality level of the sources. • **RE43**: The System to evaluate the functional aspects of the score must track the number of test cases that pass out of all test cases • **RE44**: The System to evaluate the timeliness of the score must track the time passed between the registration deadline and the last commit of a team • **RE45**: The System to compute the quality level of the sources must be able to extract through static analysis tools multiple aspects such as security, reliability, and maintainability • **RE49**: The System at the end of each battle must be able to update the personal tournament score of each student • **RE49.1**: The System must be able to calculate the personal tournament score of each student by summing of all battle scores received in that tournament
Components	Score Calculation Service, integrating Automated Evaluation Handler, Manual Evaluation Integration, and Static Analysis Tools Integration.

Table 4.4: Score Calculation and Evaluation

	GitHub Integration and Repository Management
Requirements	• **RE29**: The system must automatically create a GitHub repository for the code kata, when the deadline expires. • **RE30**: The system must automatically notify enrolled student groups of a battle about the link to the GitHub repository for the code kata, when the deadline expires. • **RE31**: The System must be able to retrieve all the information from GitHub • **RE39**: The System must update the team's rank after a student, who is part of the team, executes a commit.
Components	GitHub Service.

Table 4.5: GitHub Integration and Repository Management

	Notification Management
Requirements	• **RE10**: The System must ask a Guest permission to send notifications • **RE11**: The System must process the Guest's choice about the notification permission • **RE40**: The System must send a notification to all subscribed Students when a new tournament is created. • **RE41**: The System must send a notification to all Students enrolled in a tournament, when a new battles are created within it. • **RE46**: The System must allow Students to view their team battle score • **RE47**: The System must allow Students to view the battle team leaderboard, if and only if the students are enrolled in the battle • **RE48**: The system must notify Students when the final battle rank becomes available, if and only of the student is enrolled in that battle • **RE50**: The System must notify all Students involved all students involved in the tournament closed by an educator, as soon as the final tournament rank becomes available. • **RE51**: The System must send notifications by email, and through the platform • **RE62**: The System must have the capability to interface with an external email service provider for sending automated emails. This includes generating email requests containing appropriate content (such as notifications or invitations) and transmitting these requests securely to the email provider for distribution to the intended recipients
Components	Notification Service and Email Service Interface.

Table 4.6: Notification Management

	Team Formation and Group Management
Requirements	• **RE16**: The system must allow Students to create a team per each battle (the team is single-use per each battle) • **RE24**: The System must not allow Students to invite more friends in a battle than the maximum number required to build a team • **RE25**: The System must allow invited students to either accept or reject the invitation of joining a battle • **RE26**: The System must not allow a Student to join a closed battle • **RE27**: The System must not allow an Educator without "creator" privileges in a tournament to create a battle within it • **RE60**: The System must set to "Pending" the status of a Student's request to join a battle with a minimum group size greater than one • **RE61**: The System must set to "Accepted" the status of a Student's request to join a battle, if and only if he is part of a team that meets the min-max criteria group formation
Components	Battle Management Service, specifically the Team Formation Manager.

Table 4.7: Team Formation and Group Management

5 | Implementation, Integration, Testing

5.1. Implementation and Integration

The implementation, integration, and testing phase of the CodeKataBattle software project is characterized by a structured and systematic approach, focusing on the efficient development and rigorous testing of its components. This phase begins with the implementation of modules that directly interface with DBMSs, laying a robust foundation for the system's core functionalities. Following this, modules responsible for managing external service communications are developed, ensuring seamless interaction with various external entities.

Subsequently, the focus shifts to components that rely heavily on the services established by the initial modules. This step is crucial for maintaining system integrity and functionality, as it involves integrating components that have significant dependencies on the previously developed services. The implementation strategy here is meticulous, ensuring that each component is fully functional and seamlessly integrates with the existing system framework.

The final stage of the process involves the integration of the most complex modules. These modules are characterized by their extensive interactions with a multitude of services within the system. The implementation of these modules is the culmination of the development process, requiring a comprehensive understanding of the entire system architecture.

Throughout this phase, a parallel development approach is employed for front-end and back-end components. This strategy not only facilitates a quicker development process but also aids in the early detection and resolution of bugs, ensuring a more stable and reliable software product. The implementation plan is designed to be incremental, allowing for gradual growth and development of the system. This incremental approach is instrumental in maintaining a high level of control over the development process, ensuring that each

component is thoroughly tested and integrated before moving on to the next.

5.2. Practical Implementation Plan

As we have said previously, the implementation plan for the CodeKataBattle software project is strategically designed to prioritize the development of components in a sequence that aligns with their dependencies and criticality to the overall system functionality. The plan is laid out as follows:

5.2.1. Initial Phase: Core Service Components

The first phase of implementation focuses on the core service components that are foundational to the system and have direct interactions with the databases. This includes:

1. **User Management Service** This service is critical as it handles user registration, profile management, and authentication. It is directly connected to the User Management Database, making it essential to establish this component early in the development process.
2. **Tournament Management Service** Since this service manages the entire lifecycle of tournaments, from creation to closure, and interacts with the Tournament Database, it is pivotal to implement it at the outset. It ensures that the fundamental framework for tournament operations is in place.
3. **Battle Management Service** This service is integral to the management of battles within tournaments. Its early implementation is crucial as it interacts with the Battle Database, laying the groundwork for the battle-related functionalities.

Developing these services first ensures that the primary database-dependent functionalities are established, forming a robust backbone for the system.

5.2.2. Secondary Phase: Score Calculation and External Integrations

Once the foundational services are in place, the focus shifts to:

4. **Score Calculation Service** This service is vital for calculating scores from various sources. It needs to interact with both the **Static Analysis Tools** and the Scoring Database. Its development follows the core services to ensure seamless integration and data flow.

Concurrently, development will commence on:

5. **GitHub Service** Essential for repository and commit management, and integrating GitHub Actions.
6. **Email Service Provider** To set up the infrastructure for sending out notifications and communications.

These components are developed in parallel due to their interdependencies and their role in enhancing the functionality of the core services.

5.2.3. Final Phase: Notification Service and Web Server

The last phase of the implementation includes:

7. **Notification Service** This service is implemented towards the end as it hinges on the functionality of the earlier services for triggering notifications related to tournaments, battles, and scores.
8. **Web Server** Being the conduit through which all other components communicate with the User Interface, its implementation is reserved for the final stage. At this point, most services are already in place, allowing for comprehensive testing and integration of the Web Server with all system components.

5.3. Testing Plan

The testing plan for the CodeKataBattle software project is devised to align with the phased implementation strategy, ensuring thorough and efficient validation of each component at various stages of its development. The plan is structured to encompass a range of testing methodologies, progressively covering unit, integration, system, and user acceptance tests.

5.3.1. Phase-Wise Testing Approach

The testing activities correspond to the phased implementation approach, ensuring that each component is rigorously tested before moving to the next phase of development.

1. **Initial Phase - Core Service Components Testing**
    - *Unit Testing*: Each core service component – User Management Service, Tournament Management Service, and Battle Management Service – undergoes unit testing. This involves testing individual functions and methods for correctness.

- *Database Integration Testing*: Testing the interaction of these services with their respective databases (User Management, Tournament, and Battle Databases) to ensure data integrity and proper data flow.

2. **Secondary Phase - Score Calculation and External Integrations Testing**
   - *Integration Testing*: After the core services and the Score Calculation Service are developed, integration tests are conducted to ensure these services work cohesively.
   - *External Service Integration Testing*: Concurrently, GitHub Service and Email Service Provider are tested for their integration with the core services and their respective external systems.
   - *Score Calculation Service Testing*: This involves testing the integration with Static Analysis Tools and the Scoring Database.
3. **Final Phase - Notification Service and Web Server Testing**
   - *System Testing*: The entire system, including the Notification Service and Web Server, is subjected to system testing. This ensures that all components work together seamlessly as a single unit.
   - *Web Server Integration Testing*: Special focus is given to testing the integration of the Web Server with all other components, considering its pivotal role in handling requests and serving responses.

5.3.2. Comprehensive Testing Methodologies

- *Functional Testing*: Conducted across all phases to validate that each component and the system as a whole function according to the specified requirements.
- *Non-Functional Testing*: Includes performance, security, and usability testing. These tests are crucial to ensure the software not only works correctly but is also efficient, secure, and user-friendly.

5.3.3. User Acceptance Testing (UAT)

In the final stage of testing, User Acceptance Testing is performed. This involves real users testing the system in a real-world scenario. This step is crucial to ensure that the system meets user expectations and requirements.

5.3.4. Iterative Testing and Continuous Integration

Throughout the development process, an iterative testing approach is adopted. As new features and components are added, regression tests are conducted to ensure that new changes do not adversely affect existing functionalities.

Continuous integration practices are employed, where automated tests are run with every code commit. This helps in early detection of issues and maintaining overall code quality.

By following this comprehensive and phase-aligned testing plan, the CKB project aims to ensure that each component is not only functional in isolation but also operates effectively within the integrated system.

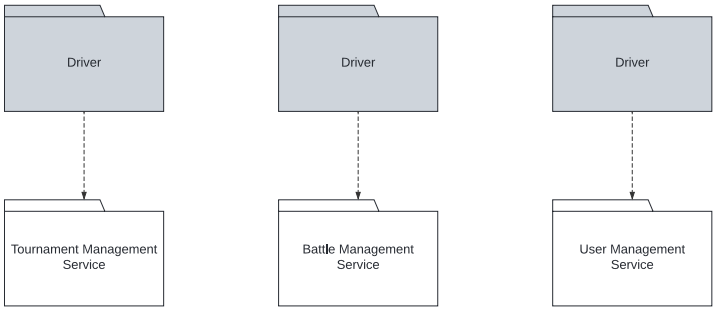

Figure 5.1: First phase of the Implementation Plan - Package Diagram

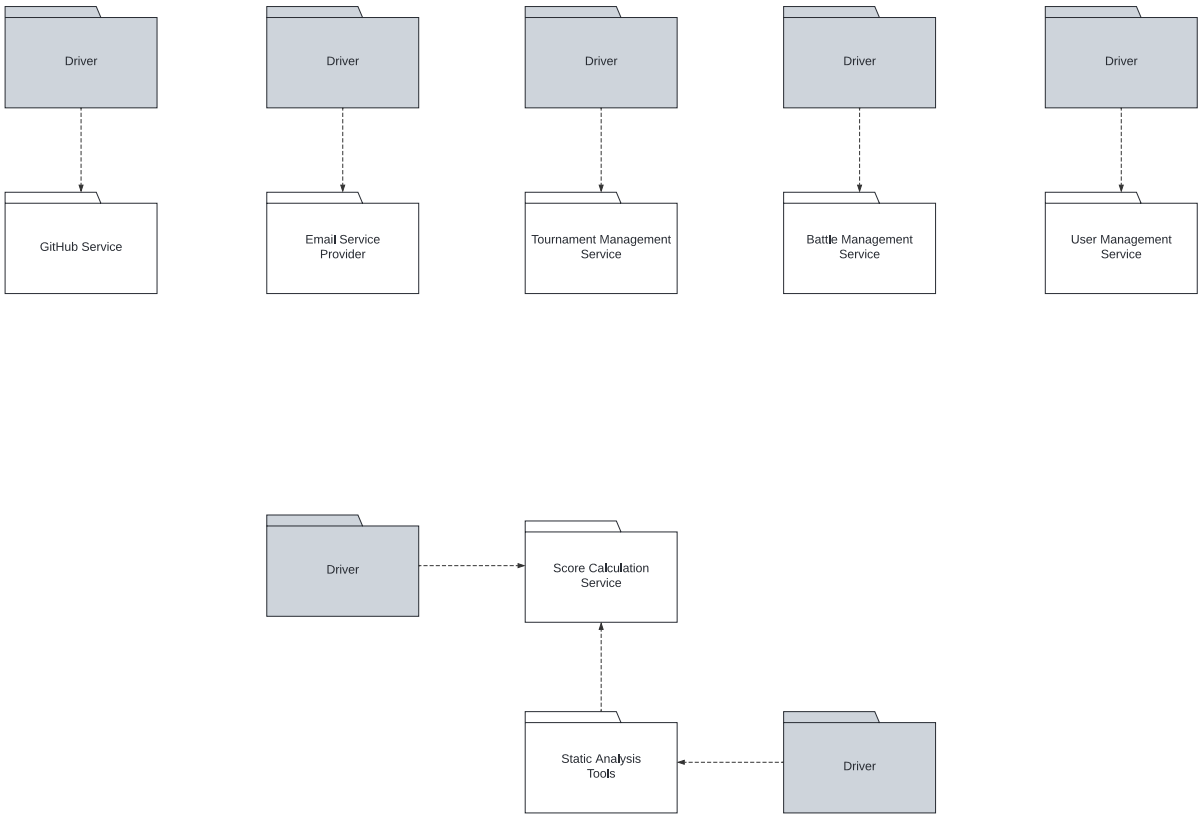

Figure 5.2: Second phase of the Implementation Plan - Package Diagram

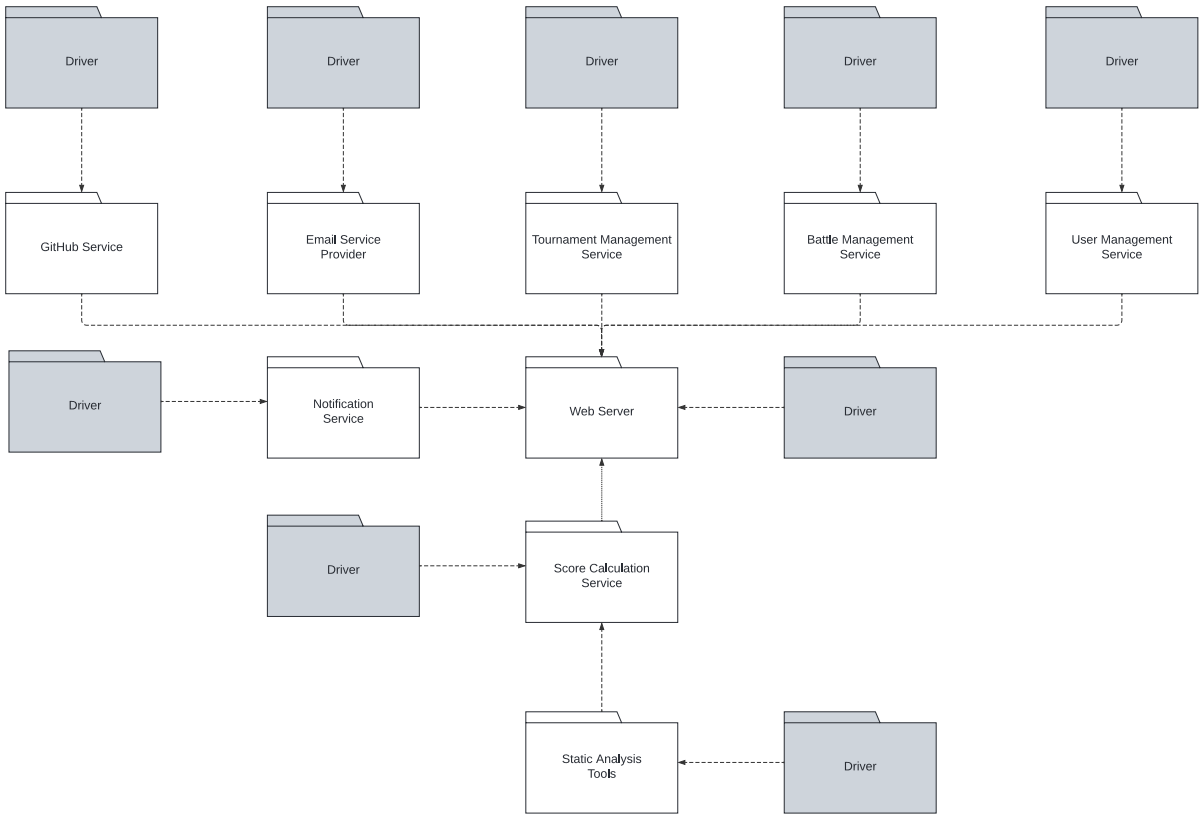

Figure 5.3: Third phase of the Implementation Plan - Package Diagram

6 Effort Spent

6.0.1. Effort-Topazio

Hours	Description
4	Discussion about the Assignment 2023-2024
4	Introduction
17	Architectural Design
4	User Interface Design
8	Requirements Traceability
2	Implementation,Integration and test plan
39	Total

Table 6.1: Effort Spent - Topazio

6.0.2. Effort-Gigante

Hours	Description
4	Discussion about the Assignment 2023-2024
2	Introduction
13	Architectural Design
10	User Interface Design
2	Requirements Traceability
8	Implementation,Integration and test plan
39	Total

Table 6.2: Effort Spect - Gigante

List of Figures

2.1 High-level description of the System's architecture 10
2.2 Component view of the System 12
2.3 Deployment View 15
2.4 User Registration 17
2.5 User Login 18
2.6 Tournament Creation 19
2.7 Tournament Enrollment 20
2.8 Battle search 21
2.9 Join Battle Individually 22
2.10 Group Formation 23
2.11 Send Battle Invitation 24
2.12 Join Battle By Invitation 25
2.13 Change Languages Settings 26
2.14 Battle Creation 27
2.15 Score & Battle Update 28
2.16 Close Tournament 29
2.17 View Battle Score 30
2.18 View Tournament Score 31
2.19 Manual Evaluation 32

3.1 Guest - Home Page 47
3.2 Guest - Sign up / Log in Interface 1 48
3.3 Guest - Sign up / Log in Interface 2 48
3.4 Educator - Sign Up Page 49
3.5 Educator - Log In Page 50
3.6 Educator - Home Page 50
3.7 Educator - Creating a New Tournament Interface 51
3.8 Educator - Insert Educators Interface 51
3.9 Educator - Confirm Tournament Creation Interface 52
3.10 Educator - Tournament successfully created 52

3.11 Educator - Personal Area Page 53
3.12 Educator - Change Settings Interface 54
3.13 Educator - Notifications Page 54
3.14 Educator - Contact Us Page 55
3.15 Educator - Available Tournaments Page 55
3.16 Educator - Tournament Page 56
3.17 Educator - Create a New Battle Interface 57
3.18 Educator - Battle successfully created 57
3.19 Educator - Battle Page 58
3.20 Educator - Project Page 59
3.21 Student - Sign Up Page 60
3.22 Student - Log In Page 61
3.23 Student - Home Page 61
3.24 Student - Personal Area Page 62
3.25 Student - Change Settings Interface 62
3.26 Student - Notifications Page 63
3.27 Educator - Contact Us Page 64
3.28 Student - Available Tournaments Page 64
3.29 Student - Tournament Page 65
3.30 Student - Battle Page 66
3.31 Student - Join Battle Interface 67
3.32 Student - Insert Students Interface 68
3.33 Student - Battle request correctly sent 68

5.1 First phase of the Implementation Plan - Package Diagram 82
5.2 Second phase of the Implementation Plan - Package Diagram 82
5.3 Third phase of the Implementation Plan - Package Diagram 83

List of Tables

1.1 4

4.1 User Account Creation and Management 70
4.2 Tournament and Battle Management - 1 71
4.3 Tournament and Battle Management - 2 72
4.4 Score Calculation and Evaluation 73
4.5 GitHub Integration and Repository Management 74
4.6 Notification Management 75
4.7 Team Formation and Group Management 76

6.1 Effort Spent - Topazio 85
6.2 Effort Spect - Gigante 85