Software Engineering 2

CodeKataBattle
Design Document
version 1.0

Francesco Galbiati, Chiara Fossà

4th January 2024

Contents

1 Introduction **3**
- 1.1 Purpose . . . 3
- 1.2 Scope . . . 3
- 1.3 Definitions, acronyms, abbreviations . . . 3
  - 1.3.1 Definitions . . . 3
  - 1.3.2 Acronyms . . . 4
- 1.4 Revision history . . . 4
- 1.5 Document structure . . . 4

2 Architectural design **6**
- 2.1 Overview . . . 6
  - 2.1.1 High level view . . . 6
  - 2.1.2 Distributed view . . . 7
- 2.2 Component view . . . 9
- 2.3 Deployment view . . . 12
- 2.4 Runtime view . . . 14
- 2.5 Component interfaces . . . 28
- 2.6 Selected architectural style and patterns . . . 34
  - 2.6.1 Client-Server Architecture . . . 34
  - 2.6.2 Three-Tiered Architecture . . . 34
  - 2.6.3 Model-View-Controller . . . 34
  - 2.6.4 Circuit Breaker . . . 35
- 2.7 Other design decisions . . . 35
  - 2.7.1 Remote Method Invocation . . . 35
  - 2.7.2 Server Replication . . . 35

3 User interface design **36**

4 Requirements traceability **42**

5 Implementation, integration and test plan **48**
- 5.1 Overview . . . 48
- 5.2 Milestones . . . 48
- 5.3 System Testing . . . 56
- 5.4 Future Development . . . 56

6 Effort spent **57**
- 6.1 Francesco Galbiati . . . 57
- 6.2 Chiara Fossà . . . 57

7 References **57**

1 Introduction

1.1 Purpose

The purpose of this DD is to provide a detailed and structured view of the system design of the CKB platform and is essential to communicate clearly and completely to developers, stakeholders and all figures involved in the software development process. The architectural choices, design models, component interfaces, and other crucial decisions made during the system design phase are illustrated within the DD. The main purpose of this document is to ensure that all members of the development team have a clear understanding of the structure of the system to facilitate both implementation, integration and testing. Furthermore, the DD demonstrates how each requirement has been satisfied in the system design, ensuring a basis for requirements traceability.

1.2 Scope

CodeKataBattle (CKB) is a software system platform designed to help students improve their software development skills through individual and collaborative training sessions: "code kata battles". Educators can use this platform to provide various coding exercises to students to encourage competition and skill improvement. A battle on CKB is based on a programming exercise that must be done in an assigned programming language (e.g. Java, Python...) and within a deadline. Each exercise is assigned by providing an assignment and requires students to create and submit a project software. The project is automatically evaluated based on a set of test cases that the student code must pass. Participants have the opportunity to organize themselves into teams and must complete the project by writing their code following a test-first approach. During the battle, teams compete with each other to deliver a solution that meets the specified requirements and passes the provided test cases based on which the platform assigns a score. When a battle ends CKB assigns scores to the submitted solutions and creates a ranking (there is also the possibility for educators to carry out a manual evaluation). These paragraphs resume how CKB platform allows students to improve their programming skills in an engaging and interactive way using a test-driven development approach.

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

- **Battle**
  Challenge in which those who compete submit a project that will be subjected to an evaluation and scored from 0 to 100. The objective of the participants is to achieve the best possible score. Each battle has its own ranking.

- **Tournament**
  Context in which educators can propose battles to the students enrolled in it and the students can participate in them. Each tournament has its own ranking made up of the rankings obtained from the battles contained in it.

- **Subscribe**
  It is a registration procedure that allows students to express their interest in participating in a tournament or battle. It occurs through the click of a button on your profile and allows the student who has signed up to access specific features.

- **Ranking**
  A list of ratings that are the result of a competition. The list associates a student's username with their score and is sorted in descending order of grade. This can be the ranking drawn up at the end of a battle or the ranking of a tournament.

1.3.2 Acronyms

- CKB: Code Kata Battle
- GDPR: General Data Protection Regulation
- UI: User Interface
- EU: European Union
- EEA: European Economic Area

1.4 Revision history

- Version 1.0: First Submit

1.5 Document structure

The current document is divided into four main chapters.

1. **Introduction:** The first chapter contains an introduction that explains the purpose of the document, after which it briefly recalls the concepts introduced in RASD. Then important information is given to the reader: definitions, acronyms, synonyms, and the total of referenced documents.
2. **Architectural Design:** This includes a detailed description of the architecture of the system, including an overview of the elements, the software components, and a runtime diagram description of the various features of the system. In the end, we provide a detailed explanation of the architectural pattern we used.

3. **User Interface Design:** A model of application user interfaces and the links between them is provided to facilitate the understanding of their flow.

4. **Requirements Traceability:** Describes the relationship between requirements defined in RASD and the components described in the first chapter. By taking into account requirements, it provides evidence that the system can meet its objectives.

5. **Implementation, Integration, and Test Plan:** Describes the implementation, integration, and testing process that developers must follow to create the right system in the right way.

6. **Effort Spent**

7. **References:** This document contains references to all documents and software used in this document.

2 Architectural design

2.1 Overview

This section provides an overview of the architectural elements that make up the system, how they interact, and the replication mechanisms chosen to distribute the system.

2.1.1 High level view

The CKB system is articulated as a three-tiered distributed system, with an as thin as possible client. In particular, the Data Tier is realized as a central database, containing the data regarding every user, battle and tournament in the system, the database is developed using MySQL, and access to it is restricted by adequate policies. The Application tier contains the logic of the application and is the component to which clients refer to access the data stored in the Data Tier. Ultimately, the Presentation Tier is the Web App that users refer to to access the CKB system, this layer is responsible for the user interface and the communication between users and the application layer. Each layer is, of course, managed by a different hardware architecture, a central data storage for the Data Tier, a group of synchronized servers for the Application Tier and the machines on which the clients run for the Presentation Tier. Communication between the different components is handled according to the RMI protocol, which is well suited for client-server communication and systems developed using an object-oriented approach since it naturally separates implementations from interfaces, which is one of the basic object-orientation principles. The CKB system has also to handle the connection with GitHub and with the Static Evaluation Tools used for the automatic scoring of the submitted solutions, in particular, the system uses the Codacy Static Analysis Tool. CKB connects to these external systems through the proper APIs, making code evaluation automatic from push to result notification. More details about these aspects are described during the course of this document.

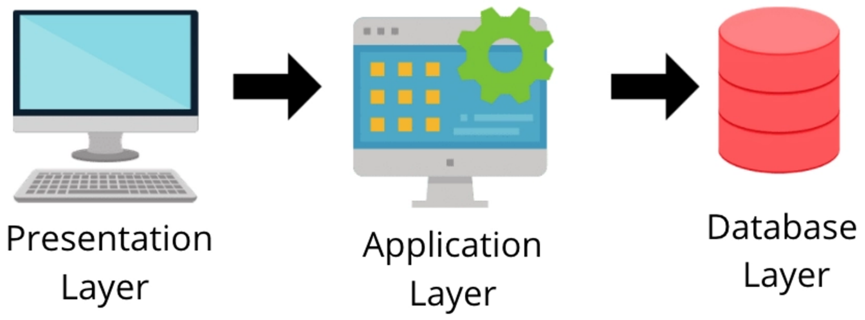


Figure 1: Three-tiered architecture diagram

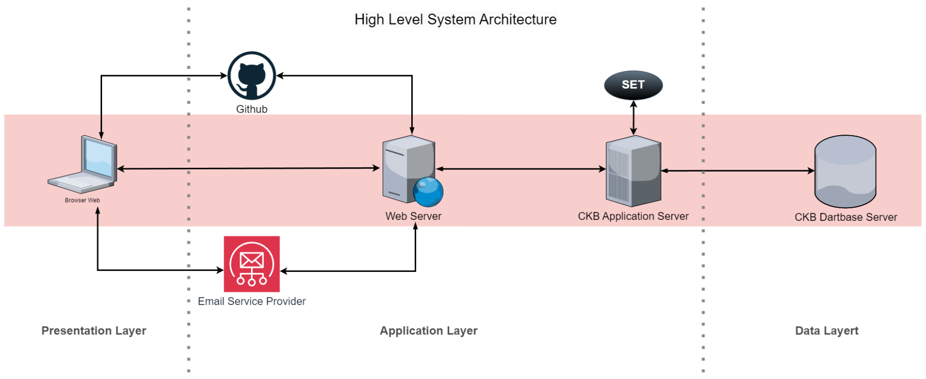


Figure 2: High level architecture of the CKB system

2.1.2 Distributed view

This section features the elements that make up the distributed architecture. The elements reported below are taken from section **??** and are represented here to be explored in more detail.

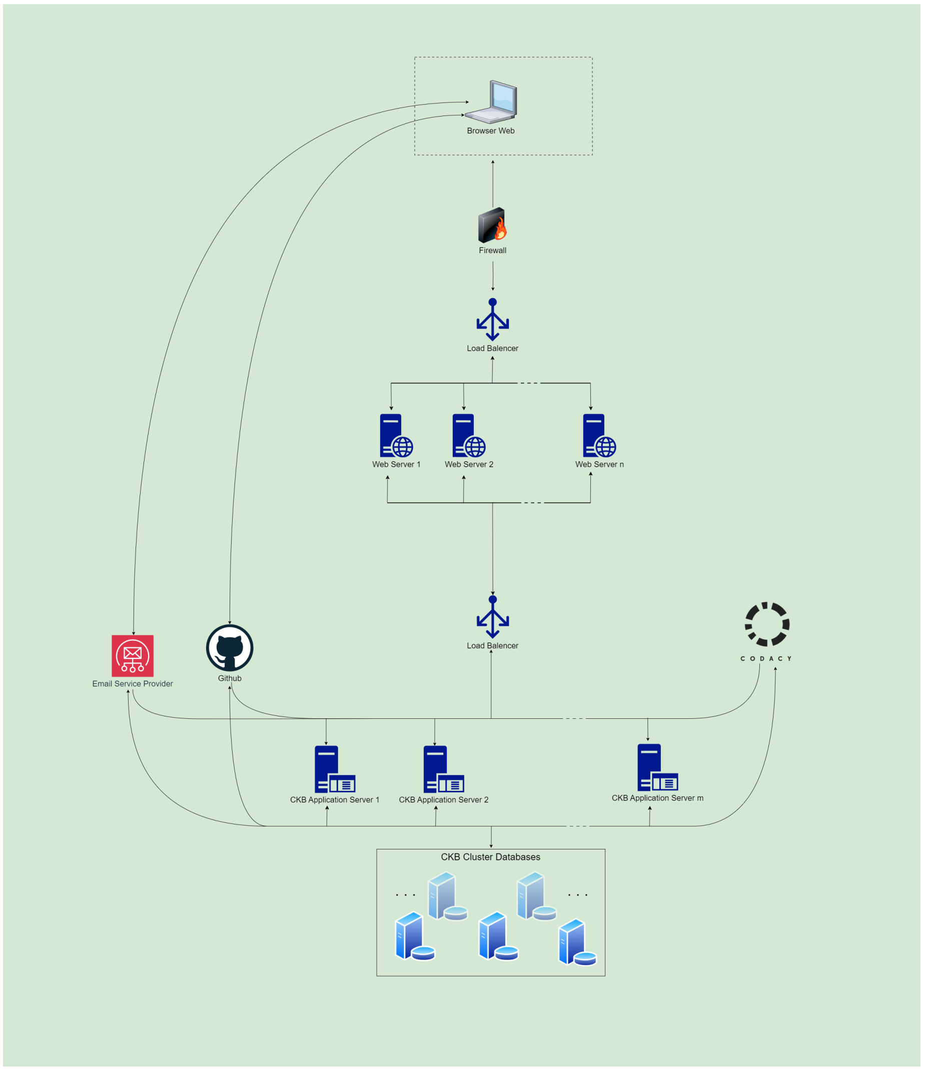


Figure 3: Distributed architecture of the CKB system

Load Balancing Two load balancers can be found in the represented architecture. Their purpose is to distribute the load of incoming requests to web servers and application servers Requests from the user directed towards the web servers arriving from the web browser, after passing the firewall, are intercepted by the first load balancer which will take care of redirecting them towards the most suitable web server able to manage them at that moment. After a request arrives at the assigned web server, it is processed, manipulated and redirected to the application servers. Before reaching an application server, the request is intercepted by the second load balancer. The second load balancer directs the requests coming from the web servers towards one of the application servers according to the criteria also respected by the first load balancer: the chosen web application will be the best available at the moment that can manipulate the request.

Application Servers The application tier processes all dynamic content and the interactions between the presentation and data tier. The application tier performs as a variety of dynamic gateways. The information that comes from the presentation tier, where it was collected, is processed in the application tier according to precise rules defined in the business logic. In addition to processing the information and comparing it with other data in the data tier, the application can communicate with the data tier, and interact, modify, add or delete the data present in it.

Clustered DBMS Using a cluster DBMS offers several advantages. A clustered system guarantees greater reliability and availability of data since the presence of multiple nodes allows the load to be distributed and any failures or interruptions to be automatically managed. This ensures greater system resilience by reducing the risk of losing critical data. The use of a cluster DBMS, then, can improve application performance by allowing the intelligent distribution of requests between nodes. Finally, a DBMS cluster provides the flexibility to scale out storage and compute resources.

2.2 Component view

In this section the document reports the component diagram of the CKB system, providing a brief description of each component present in the diagram. For the sake of readability components with different functions are highlighted in different colours, in particular:

- In dark purple are represented all the modules present in the Application Server, these being the Model and the different ”managers” that compose the Controller, whose aim is to manage a specific class of computations.
- In red are highlighted the third-party components to which the CKB system connects to provide certain functions to the user, like GitHub for repository management.

- In pink is represented the Web App, which is the primary method to access the platform by the users.
- The Web App relies on the Web Server (in purple) to provide the static components of the web pages.

Below the following picture, the document will provide additional insight about the different components of the diagram.

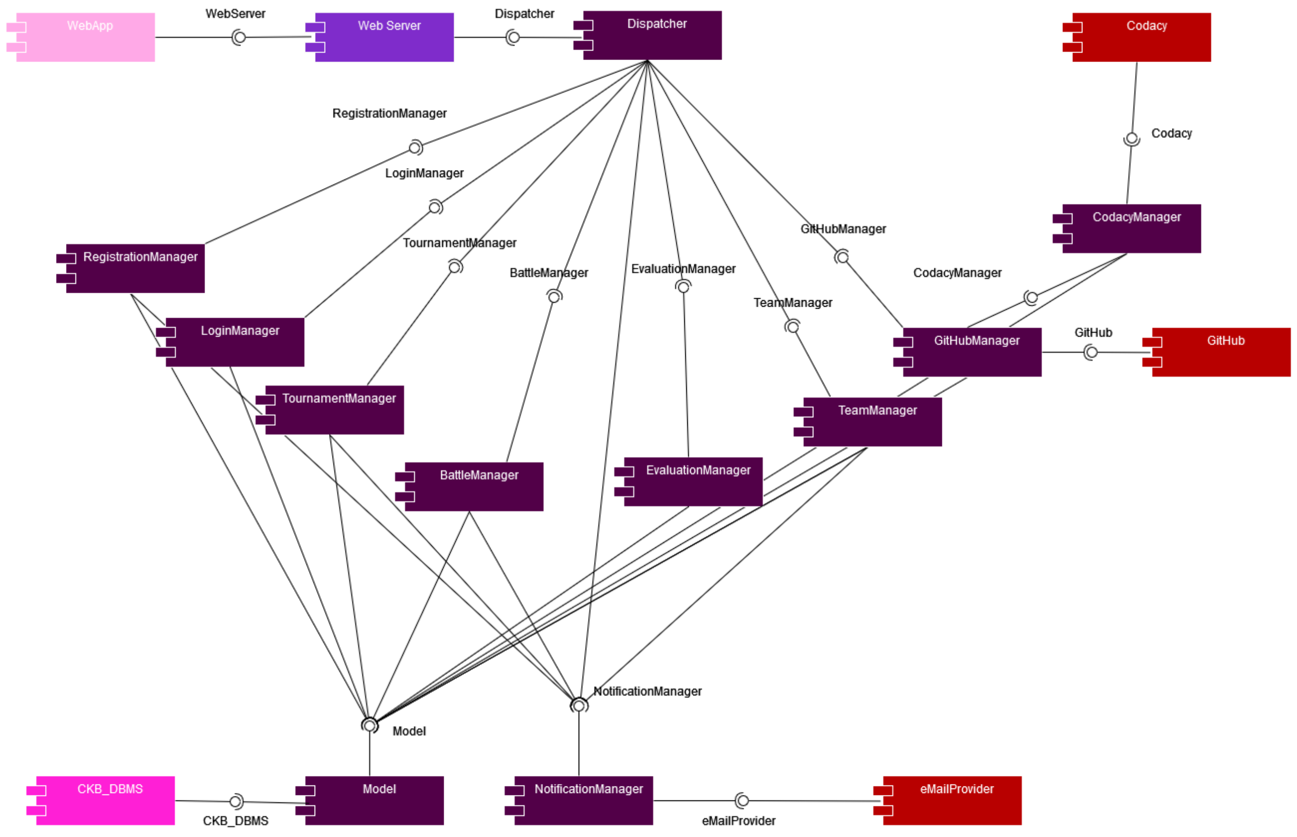


Figure 4: Component diagram of the CKB system

- The **Web App** is the component designed to provide the interface through which the clients will request the services provided by the CKB platform. It communicates with the system, in particular with the **Web Server** using the HTTP protocol.
- The **Web Server** is the component that accepts HTTP requests from the clients connected to the platform, it mainly handles web page requests, fetching them from the **Application server** and returning them to the **Web App**.
- The **Dispatcher** component is responsible for the correct forwarding of the requests coming from the **Web Server**, it ensures that the various requests are forwarded to the correct component. It also handles the return of the answers coming from the **Application Server** to the **Web Server**.
- The **RegistrationManager** component is responsible for handling the requests coming from clients that want to register an account to the CKB system and contains the methods that allow it to perform such tasks. It also has access to the **Model** to write data about new accounts.
- The **LoginManager** is responsible for the login functionalities of the CKB system, requests are forwarded to it by the **Dispatcher** and executed on the **Model**.
- The **TournamentManager** component is the part of the Application Server that can manage and provide information about existing tournaments and create new ones in the **Model**, as in the previous modules it receives requests by the dispatcher and can access to the **Model** to compute the answers to those requests. The **TournamentManager** is also responsible for the subscription process of the students to a tournament.
- The **BattleManager** handles requests about battles, mainly coming from the Educators. The "subscription to a battle" part is managed by the **TeamManager** and it is a more articulated process than the subscription to a tournament, which is handled by the **TournamentManager**. The **BattleManager** is responsible also for the consultation of battle's information and rankings.
- The **EvaluationManager** is the component responsible for, and only for, the manual evaluation phase at the end of the battles in which it is specified, it allows the Educators to consult the final solution proposed by the various teams and to assign a final score to them. The component then automatically updates the results in the **Model** and requests the **NotificationManager** to notify the participants.
- The **TeamManager** is the component responsible for the subscription of teams to a battle, in particular, it handles the invitations and automatically subscribes full teams, during the subscription phase, and teams with

at least the minimum amount of members when the registration deadline is reached.

- The **GitHubManager** handles the connection between the CKB system and GitHub, particularly the repository creation and code fetching parts. This component is also connected to the **CodacyManager**, to automatically connect to the evaluation tools when a team submits code on GitHub.
- The **CodacyManager** is responsible for the connection with Codacy, the automatic evaluation tool used by the CKB system. Requests to this module come from the GitHubManager and contain solutions to evaluate accordingly to the parameters of the specific battles. The **CodacyManager** then returns the results of the evaluation to the **Model**.
- The **NotificationManager** is the component that handles the sending of notifications to CKB users, this module accepts requests coming from various components of the Application Server and sends notifications, using the e-mail provided by each user, regarding different events that are relevant to the users (tournament creation, start of a battle, etc.).
- The **Model** is the component of the system that is meant to provide the requested data to the components of the Web Application that request it. This module also provides methods to access said data and to manipulate the various kinds of information contained in it. When requested, data is retrieved from the **CKB_DBMS**, and loaded into the **Model**, where it can be read and eventually updated. The **Model** also has to update the **CKB_DBMS** data is altered.
- The **CKB_DBMS** is responsible for the managing of the database on which the CKB system relies, its only way to communicate with the Application server is through the **Model**, from which it can receive queries and to which it returns the resulting tables, that can be loaded in the data structures contained in the model. The **CKB_DBMS** also contains personal information about the users of the system, so access to it has to be restricted.

2.3 Deployment view

The deployment view of the CKB platform is covered in this chapter. The components that characterize the hardware structure of the platform software system and the system execution environment are represented here.

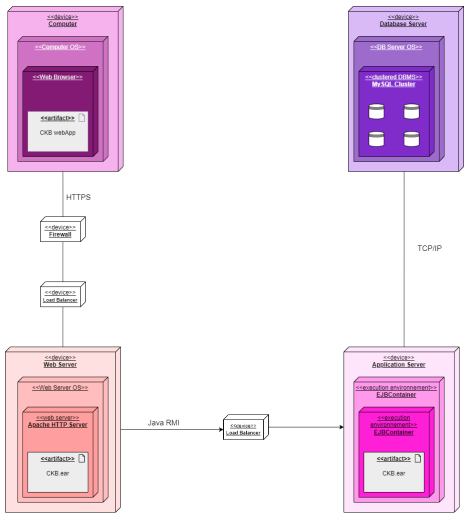


Figure 5: Deployment view of the CKB system

- **Firewall** A firewall serves as a safeguard for a private internal network against external networks, like the Internet. Operating as either a physical device or software, its main role is to monitor, filter, and regulate incoming and outgoing network traffic according to predetermined security protocols. This protection mechanism effectively prevents unauthorized entry and guarantees the privacy, authenticity, and accessibility of the network. With the ability to function across multiple layers of the OSI model, a firewall examines data packets and decides whether to permit or deny them to pass it based on predefined security policies.

- **Load Balancer** A load balancer distributes the network traffic across multiple servers preventing overload and optimizing resource usage for enhance the system performance and fault tolerance. It improves scalability, response time and ensures high availability of the web application in the distributed environment.

- **Web Server** The web server receives all the requests that the user sends to the system from the web browser. In turn, he redirects the requests to the application server via Java RMI and also returns the responses coming from the application server to the web browser. It is the web server's job to create the web page, compose it with the information that the application server provides it and present it to the user. If the user requests a page that does not require the request of information from the application server (static page, e.g. Forms), the web server returns the requested page without consulting the application server.

- **Application Server** The application server communicates with both the database and the web server. Communication with the database occurs through TCP/IP, while communication with the web server occurs through Java RMI. The application server takes care of data processing and management.

- **Database Server** The database server consists of a cluster of databases managed with Percona XtraDB Cluster, which allows synchronous and virtual replication of data on all databases in the cluster. The advantages of this choice include high availability, data integrity guarantee, possibility of horizontal scalability, automatic recovery, real-time updates, simplified management and a consistent performance.

2.4 Runtime view

In this section, the document provides the run-time view for each of the use cases defined in the RASD through the means of UML sequence diagrams. The sequence diagrams describe the interaction between the components defined in the 2.2 in the provided use cases.

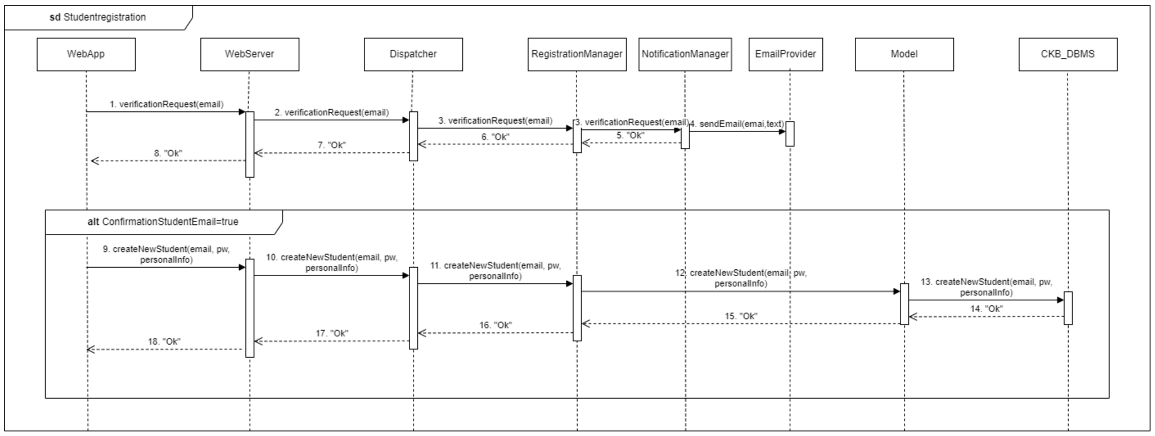

sd Studentregistration
WebApp
WebServer
Dispatcher
RegistrationManager
NotificationManager
EmailProvider
Model
CKB_DBMS
1. verificationRequest(email)
2. verificationRequest(email)
3. verificationRequest(email)
3. verificationRequest(emai
4. sendEmail(emai,text)
5. "Ok"
6. "Ok"
7. "Ok"
8. "Ok"
alt ConfirmationStudentEmail=true
9. createNewStudent(email, pw, personalInfo)
10. createNewStudent(email, pw, personalInfo)
11. createNewStudent(email, pw, personalInfo)
12. createNewStudent(email, pw, personalInfo)
13. createNewStudent(email, pw, personalInfo)
14. "Ok"
15. "Ok"
16. "Ok"
17. "Ok"
18. "Ok"

Figure 6: Runtime view of [**UC1.**]

Student registration This sequence diagram represents the registration procedure of a user who registers as a student. When a student registers on the platform through the appropriate button they are asked to verify the email provided. Once confirmation of the correctness of the email provided has been received, the student can proceed with registration by providing their requested personal data. The procedure takes place through the RegistrationManager while the NotificationManager takes care of the email verification request.

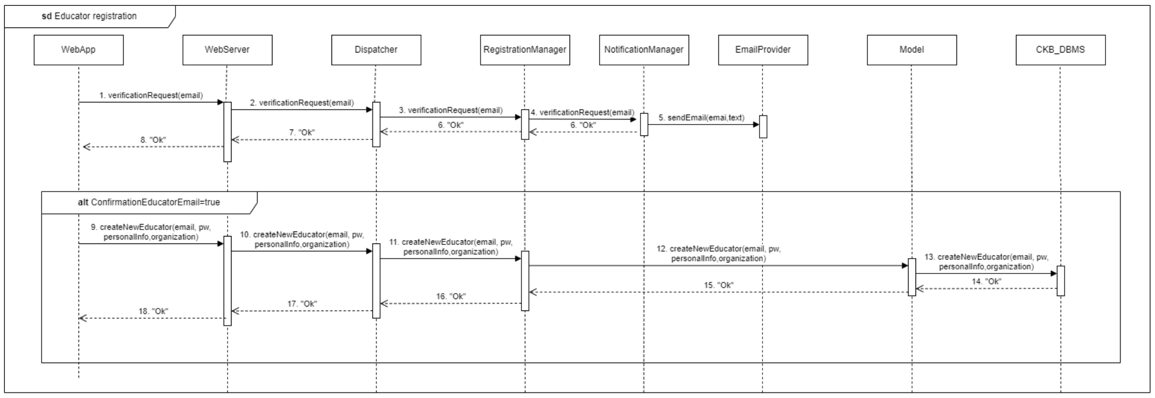

sd Educator registration
WebApp
WebServer
Dispatcher
RegistrationManager
NotificationManager
EmailProvider
Model
CKB_DBMS
1. verificationRequest(email)
2. verificationRequest(email)
3. verificationRequest(email)
4. verificationRequest(email)
5. sendEmail(emai,text)
6. "Ok"
6. "Ok"
7. "Ok"
8. "Ok"
alt ConfirmationEducatorEmail=true
9. createNewEducator(email, pw, personalInfo,organization)
10. createNewEducator(email, pw, personalInfo,organization)
11. createNewEducator(email, pw, personalInfo,organization)
12. createNewEducator(email, pw, personalInfo,organization)
13. createNewEducator(email, pw, personalInfo,organization)
14. "Ok"
15. "Ok"
16. "Ok"
17. "Ok"
18. "Ok"

Figure 7: Runtime view of [**UC2.**]

Educator registration The sequence diagram representing the educator's registration on the platform is completely similar to the registration procedure already seen for student registration. The user who intends to register as an educator will do so via the "register as an educator" button. They will then be asked to provide an email address and verify it. To complete the procedure, the educator must provide the requested personal data.

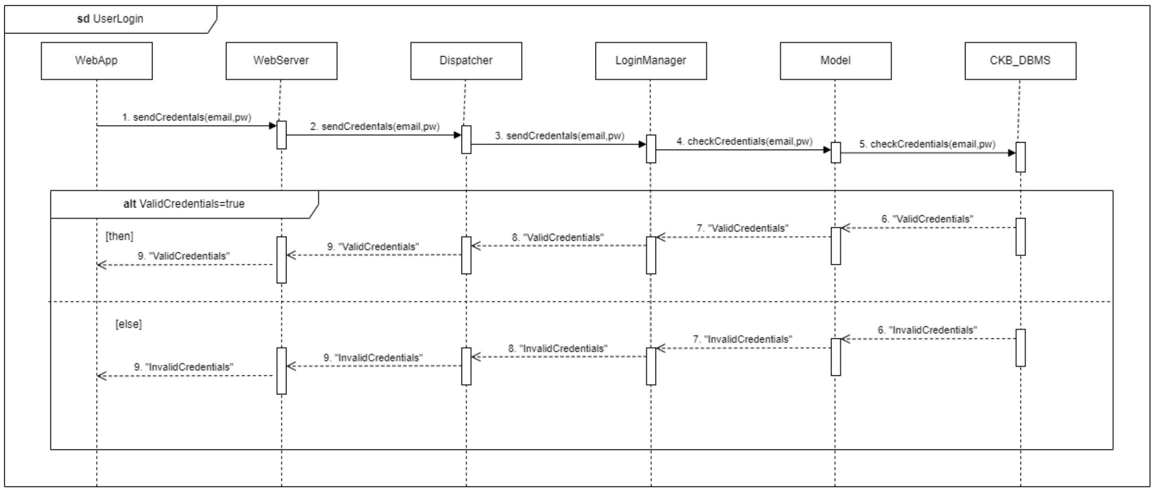


Figure 8: Runtime view of [**UC3.**]

User Login The login procedure is the same whether it is a student or an educator: the system will then recognize what type of user it is. The user who wants to log in will have to provide the email with which he registered on the platform and the password (”pw” in the diagram) to access. The procedure is managed by the LoginManager.

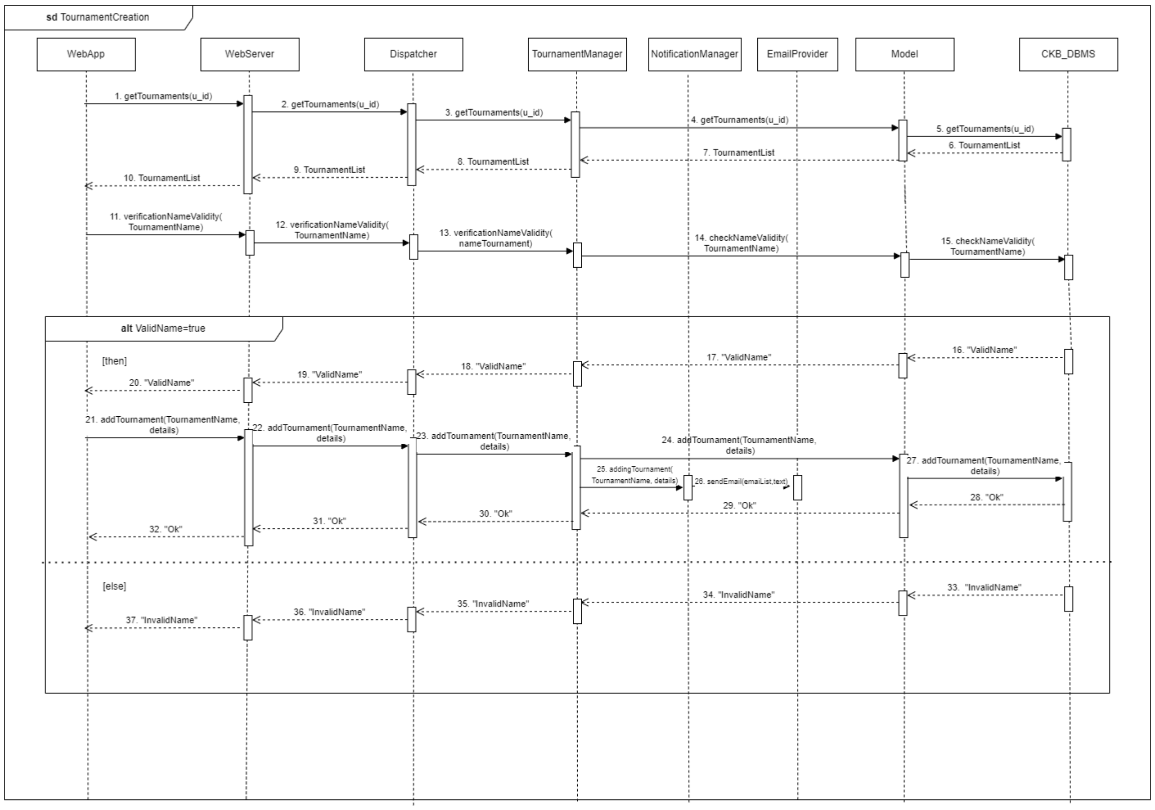


Figure 9: Runtime view of [**UC4.**]

Tournament creation This sequence diagram represents the creation of a new tournament. The educator who wants to create a new tournament will go to the MyTournaments page of his profile on the platform where he will see his existing tournaments and click on the button to add a new tournament. The educator must provide the name of the new tournament and the system verifies its validity. An educator cannot create two tournaments with the same name. If the name entered is invalid, the system indicates it. If the name provided is valid then the system informs the user of this and creates the new tournament. This operation is managed by the TournamentManager which, in the event of success in creating the tournament, requests the NotificationManager to send emails announcing the new tournament.

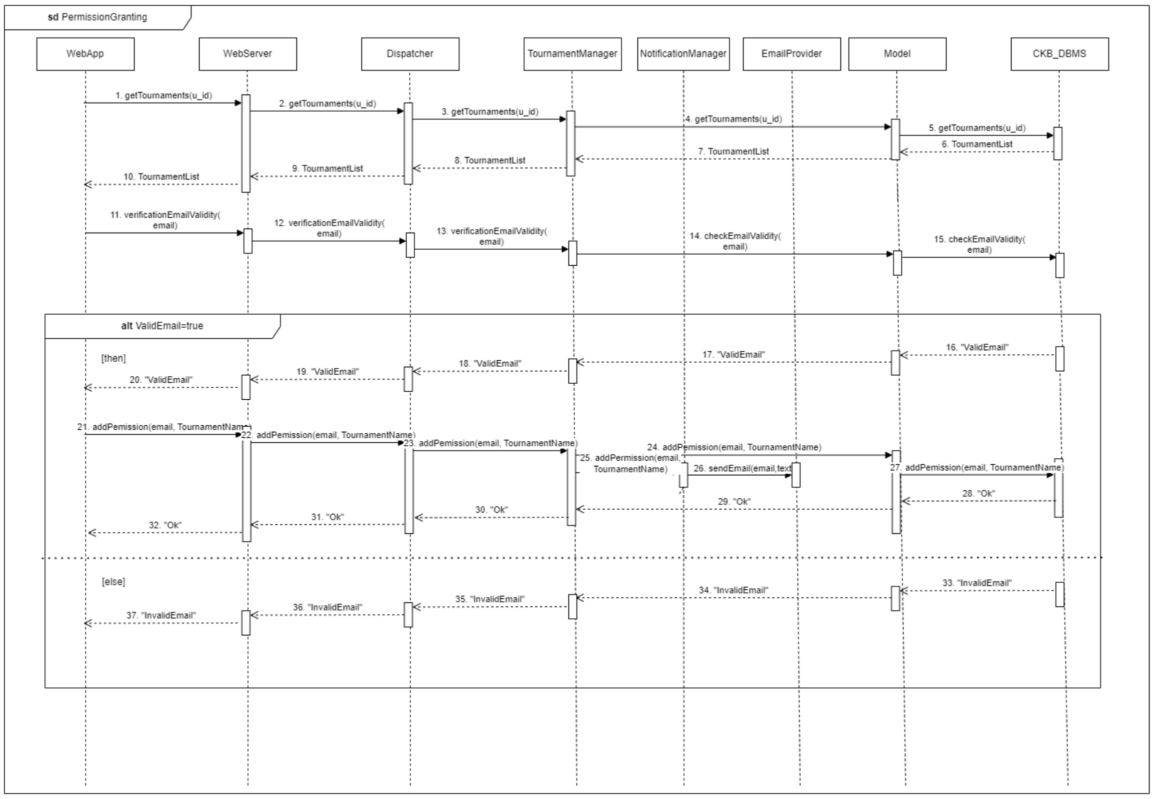


Figure 10: Runtime view of [**UC5.**]

PermissionsGranting This sequence diagram illustrates the permission-granting procedure of an educator for a tournament. The educator who created the tournament accesses her MyTournaments page (all her tournaments are shown to her) and, having selected the tournament she wants to operate on, selects the email address of the educator to whom she wants to extend permissions. The email is checked by the system. If the system identifies the email as invalid then the user is informed. If the email is valid then the system gives the green light to add the educator permission requested by the creator. Through the TournamentManager the new permit is stored in the database. At the same time, however, the TournamentManager asks the NotificationManager to send an email to the educator whose permission has just been extended to inform him.

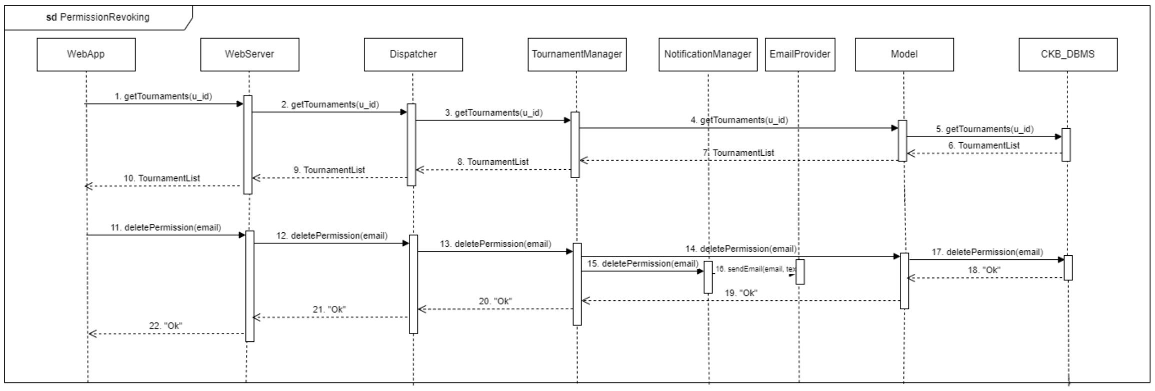


Figure 11: Runtime view of [**UC6.**]

PermissionsRevoking The sequence diagram represents the revocation of the managing permissions in a tournament by the tournament creator, unlike the addition of the permissions, revoking them does not require verification of the email. When the educator selects the tournament she wants to operate on from her MyTournaments page and clicks on the Manage Permissions button, she views the list of educators who have already been granted permission. At that point, she selects the educator she wants to remove. Through the TournamentManger the change made is transmitted to the database and through the NotificationManager the removed educator is notified.

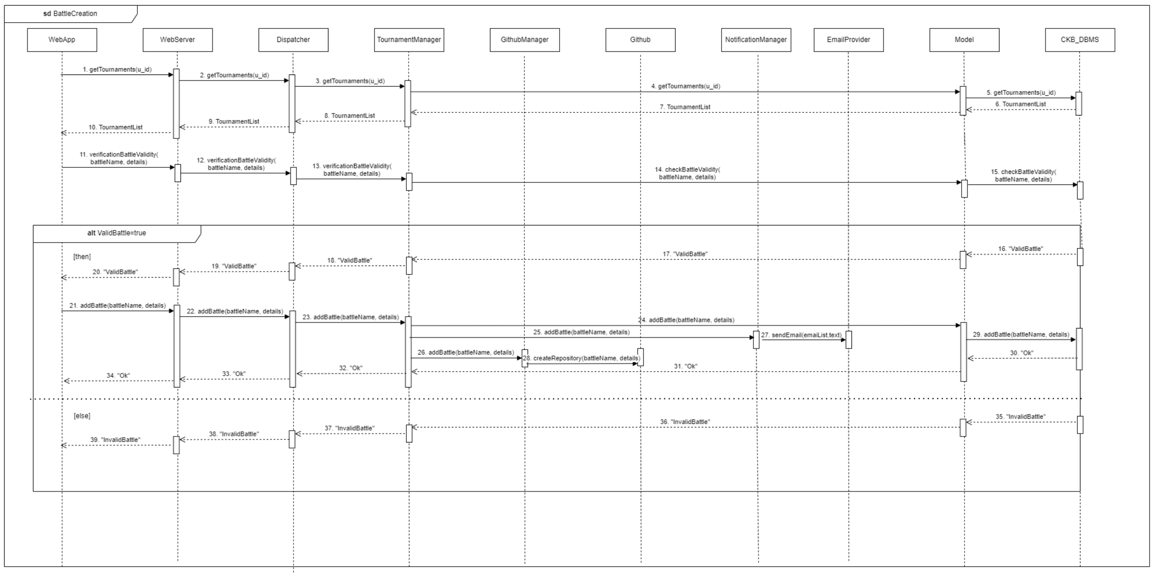


Figure 12: Runtime view of [**UC7.**]

Battle Creation This diagram shows the creation of a battle. The platform requests information on all the educator's tournaments from the database to show them to him when he accesses his MyTournaments page. The information is requested and provided through the TournamentManager. Once the desired tournament has been selected, the educator provides the data required to create the new battle. This data is propagated through the TournamentManager to the CKB-DBMS to be verified for validity. If the details provided are not considered valid by the system, the battle cannot be created and the educator is informed. (An example of one of the main constraints that a battle must meet is that there cannot be two battles with the same name in the same tournament.) If the system considers the information provided valid then it allows you to enter further details and create the new battle thanks to the TournamentManger requests the database to insert the new battle as a component of the tournament. Simultaneously with the creation of the new battle in the database, the TournamentManager requests the NotificationManager to send emails to all authorized users to inform them of the creation of the new battle. Furthermore, with the creation of the new battle the corresponding Github repository is also created through the GithubManager.

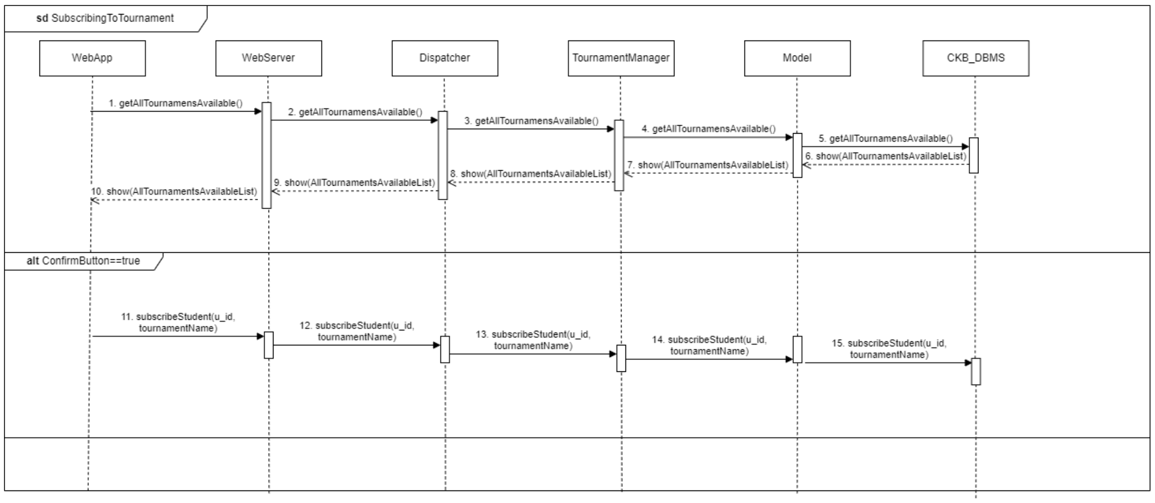

sd SubscribingToTournament
WebApp
WebServer
Dispatcher
TournamentManager
Model
CKB_DBMS
1. getAllTournamensAvailable()
2. getAllTournamensAvailable()
3. getAllTournamensAvailable()
4. getAllTournamensAvailable()
5. getAllTournamensAvailable()
6. show(AllTournamentsAvailableList)
7. show(AllTournamentsAvailableList)
8. show(AllTournamentsAvailableList)
9. show(AllTournamentsAvailableList)
10. show(AllTournamentsAvailableList)
alt ConfirmButton==true
11. subscribeStudent(u_id, tournamentName)
12. subscribeStudent(u_id, tournamentName)
13. subscribeStudent(u_id, tournamentName)
14. subscribeStudent(u_id, tournamentName)
15. subscribeStudent(u_id, tournamentName)

Figure 13: Runtime view of [**UC8.**]

Subscription to a tournament The sequence diagram showing the process of a student subscribing to a tournament is very simple. Through the TournamentManager, all the tournaments in which it is possible to register are provided to the student who requests them (this request is sent when the student enters the AllTournaments page on his profile on the platform). The student simply has to select the desired tournament and press the button to confirm his subscription. At this point, through the TournamentManager, the student's subscription is communicated to the DB.

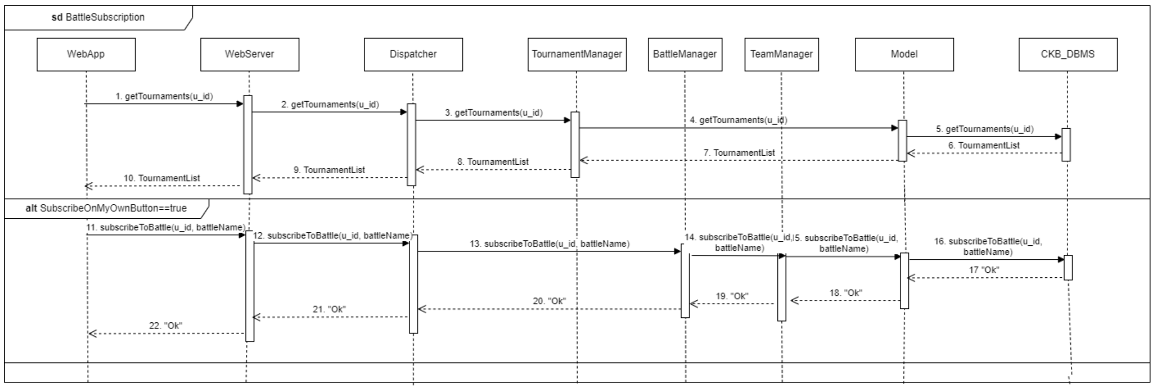


Figure 14: Runtime view of [**UC9.**]

Battle subscription This sequence diagram shows the process of an individual student subscribing to a battle. The TournamentManager, BattleManager, and TeamManager are involved in this procedure. As in previous cases, the user's access to the MyTournaments page activates the platform's request to the database of all information regarding the user's tournaments. Through the TournamentManager the request is propagated to the DB which provides all the data. Once the tournaments are displayed, the user selects a tournament and clicks on the button that allows him to register for the battle in which she wants to participate. At the click of the button the student is subscribed for the battle through the BattleManager and the TeamManager is informed of the need to create a team made up of a single user. The subscription process ends with positive feedback being provided to the user.

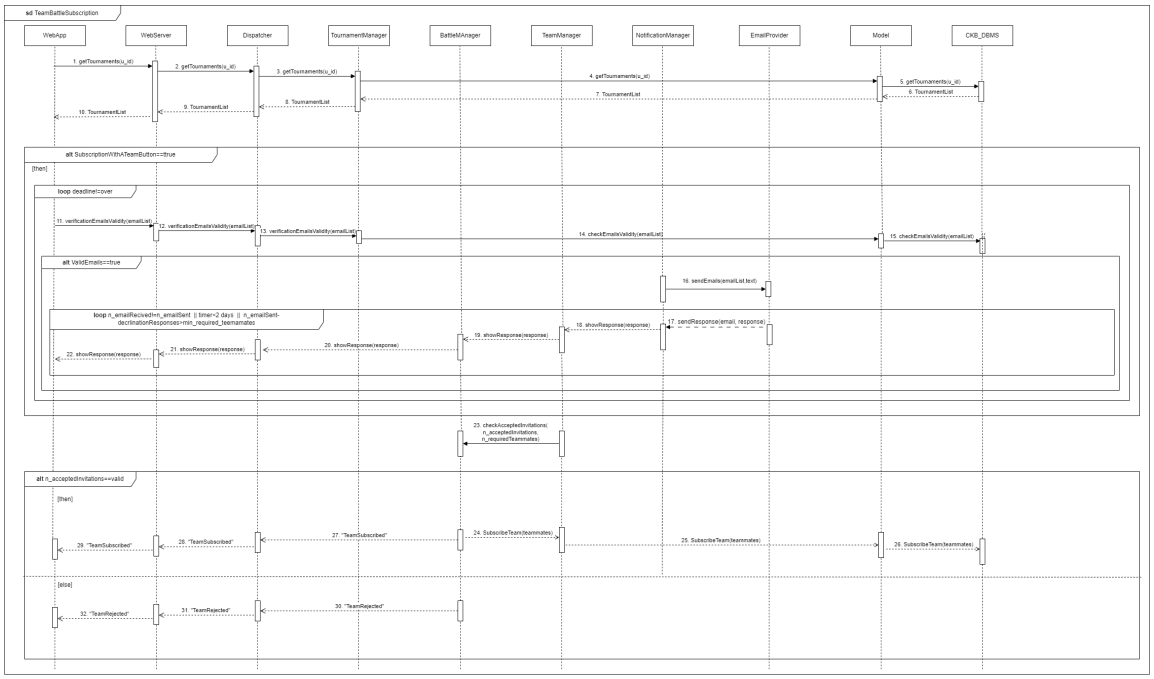

sd TeamBattleSubscription
WebApp
WebServer
Dispatcher
TournamentManager
BattleMAnager
TeamManager
NotificationManager
EmailProvider
Model
CKB_DBMS
1. getTournaments(u_id)
2. getTournaments(u_id)
3. getTournaments(u_id)
4. getTournaments(u_id)
5. getTournaments(u_id)
6. TournamentList
7. TournamentList
8. TournamentList
9. TournamentList
10. TournamentList
alt SubscriptionWithATeamButton==ttrue
[then]
loop deadline!=over
11. verificationEmailsValidity(emailList)
12. verificationEmailsValidity(emailList)
13. verificationEmailsValidity(emailList)
14. checkEmailsValidity(emailList)
15. checkEmailsValidity(emailList)
alt ValidEmails==true
16. sendEmails(emailList,text)
loop n_emailRecived!=n_emailSent || timer<2 days || n_emailSent-decrlinationResponses>min_required_teemamates
17. sendResponse(email, response)
18. showResponse(response)
19. showResponse(response)
20. showResponse(response)
21. showResponse(response)
22. showResponse(response)
23. checkAcceptedInvitations(n_acceptedInvitations, n_requiredTeammates)
alt n_acceptedInvitations==valid
[then]
24. SubscribeTeam(teammates)
25. SubscribeTeam(teammates)
26. SubscribeTeam(teammates)
27. "TeamSubscribed"
28. "TeamSubscribed"
29. "TeamSubscribed"
[else]
30. "TeamRejected"
31. "TeamRejected"
32. "TeamRejected"

Figure 15: Runtime view of [**UC10.**]

Team Subscription This sequence diagram represents the process of subscribing a team to a battle. First of all, thanks to the TournamentManager, the data of all tournaments is provided to the student. At this point, the student selects a tournament and a battle from that tournament in which he wants to enter the team. He decides to subscribe to the team tournament by clicking on a specific button. As long as the deadline has not expired, the student can provide a list of emails (corresponding to other students) to create a team. Once the emails have been checked, the system sends an invitation to all invited students, via the NotificationManger. Each response received from the guests is shown to the student "team creator". When the deadline expires, the system examines the composition of the team. If a sufficient number of people have agreed to participate in the team then the team is subscribed in the database thanks to TeamManager and the user is informed of the success of the subscription. If the number of participants does not satisfy the requests, the user will be informed of the failure of the subscription.

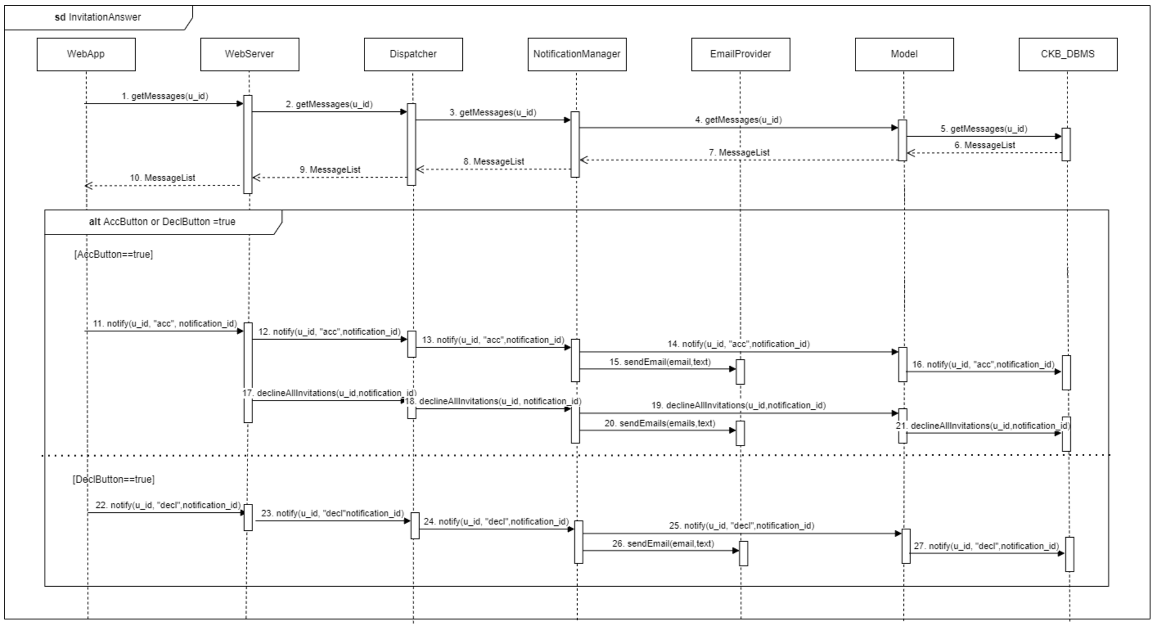

sd InvitationAnswer
WebApp
WebServer
Dispatcher
NotificationManager
EmailProvider
Model
CKB_DBMS
1. getMessages(u_id)
2. getMessages(u_id)
3. getMessages(u_id)
4. getMessages(u_id)
5. getMessages(u_id)
6. MessageList
7. MessageList
8. MessageList
9. MessageList
10. MessageList
alt AccButton or DeclButton =true
[AccButton==true]
11. notify(u_id, "acc", notification_id)
12. notify(u_id, "acc",notification_id)
13. notify(u_id, "acc",notification_id)
14. notify(u_id, "acc",notification_id)
15. sendEmail(email,text)
16. notify(u_id, "acc",notification_id)
17. declineAllInvitations(u_id,notification_id)
18. declineAllInvitations(u_id, notification_id)
19. declineAllInvitations(u_id,notification_id)
20. sendEmails(emails,text)
21. declineAllInvitations(u_id,notification_id)
[DeclButton==true]
22. notify(u_id, "decl",notification_id)
23. notify(u_id, "decl"notification_id)
24. notify(u_id, "decl",notification_id)
25. notify(u_id, "decl",notification_id)
26. sendEmail(email,text)
27. notify(u_id, "decl",notification_id)

Figure 16: Runtime view of [**UC11.**]

InvitiationAnswer This sequence diagram illustrates the procedure for responding to an invitation. An invitation can be accepted or rejected and to do so the user accesses the MyMessages section of their UI. The platform requests all the messages to be shown to the user from the DB. This request is handled by the NotificationManager. Once the message to which the user wants to respond is displayed, he can click on "Accept" or "Decline". With these options, the user will communicate whether they wish to accept or decline the invitation received respectively. By clicking on one of the two buttons the decision is recorded in the system and the parties involved are notified.

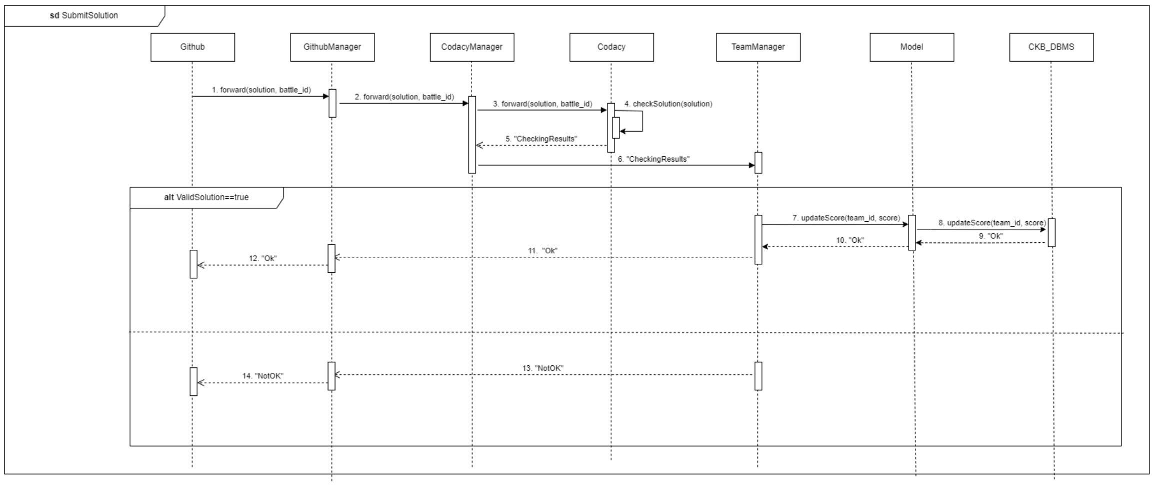


Figure 17: Runtime view of [**UC12.**]

SubmitSolution This sequence diagram illustrates the procedure for submitting a solution. The student submits the solution via GitHub. The GitHub Manager takes charge of the solution and redirects it to the Codacy which will evaluate it from the SET. The evaluation results are returned to the Codacy-Manager which redirects them to the TeamManager. At this point, if the results provided communicate that there were no problems of any kind, the score provided is invited to the database to update the team's score, and the success of the operation is propagated to GitHub and the student. In case of problems detected by SET, this circumstance is reported to the student.

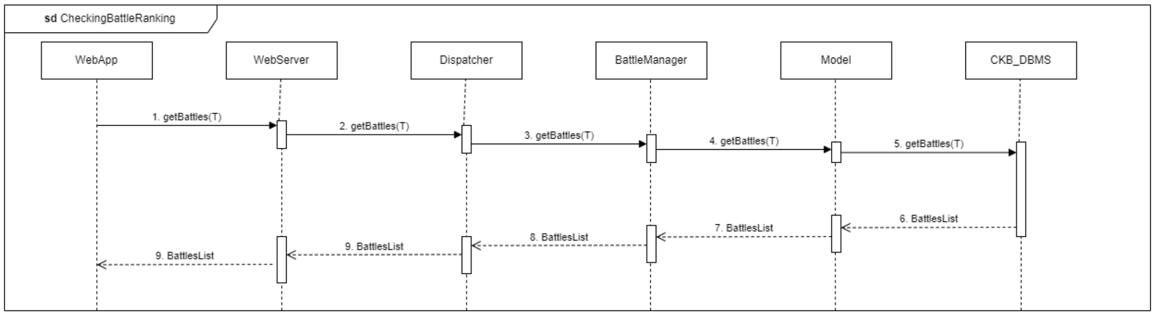


Figure 18: Runtime view of [**UC13.**]

Battle Ranking The BattleManager is involved in viewing the rankings of a battle and acts as an intermediary in the request and response of all the information regarding the battles that is requested when the user accesses the MyBattles page. Rankings are also included in the information provided.

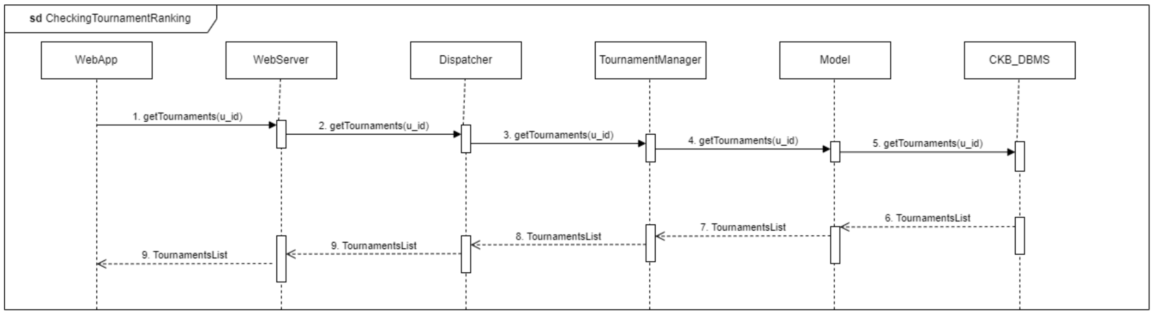


Figure 19: Runtime view of [**UC14.**]

Tournament Ranking The TournamentManager is involved in viewing the rankings of a tournament and acts as an intermediary in the request and response of all the information regarding the tournaments that is requested when the user accesses the MyTournaments page. Rankings are also included in the information provided.

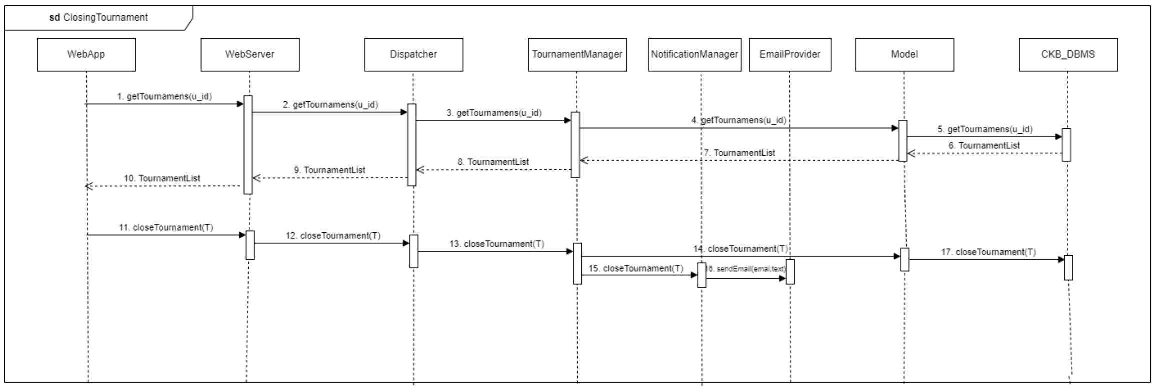


Figure 20: Runtime view of [**UC15.**]

Closing Tournament This sequence diagram shows the process of closing a tournament. The educator who wishes to close a tournament accesses the MyTournaments page of his profile. A request for information regarding the tournaments belonging to that specific user is sent. Among the information provided regarding tournaments, there is also that which specifies whether the platform must show the user, for each tournament, the button that allows it to be closed. The educator can select a tournament and if possible, click on the button to close it. When you click on the button, the request is propagated to the database through the TournamentManager, and the tournament is closed. At the same time, the NotificationManager informs all users involved of the closure of the tournament.

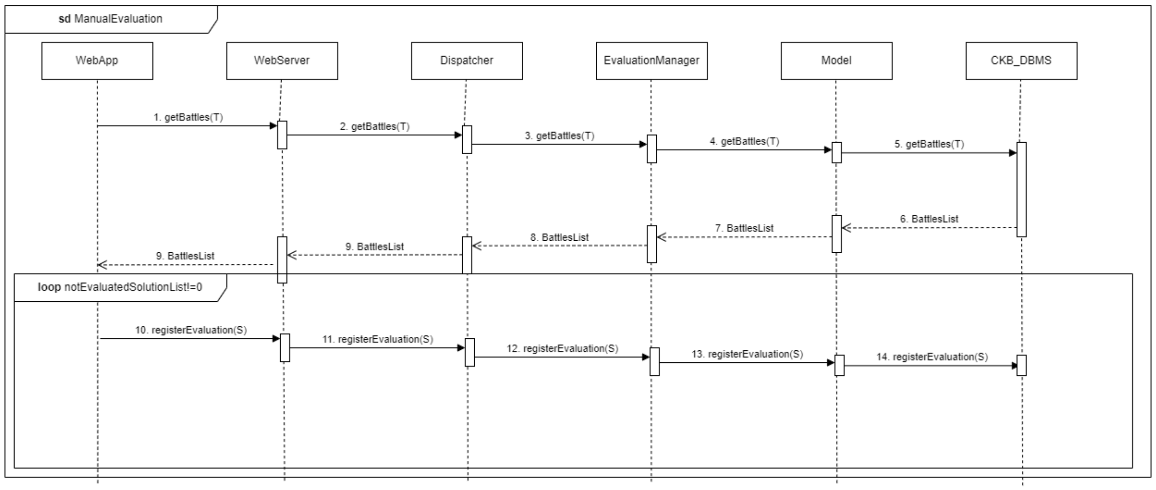


Figure 21: Runtime view of [**UC16.**]

Manual Evaluation The educator who wants to manually evaluate a completed battle accesses the MyBattles page of his profile and, through the BattleManager, he is provided with all the information regarding the battles. This information, for each battle, includes information indicating whether it is possible to evaluate the solutions manually and, if so, the list of solutions. Until all the solutions have been evaluated by the educator, they are proposed one by one to the educator who must assign an evaluation to each one.

2.5 Component interfaces

This section describes the interfaces made available by each one of the components described in 2.2, according to the diagrams described in the 2.4.

Web Server

- `verificationRequest(email)`
- `createNewStudent(email, pw, personalInfo)`
- `createNewEducator(email, pw, personalInfo, organization)`
- `sendCredentials(email, pw)`
- `getTournaments(u_id)`
- `verificationNameValidity(tournamentName)`
- `addTournament(tournamentName, details)`

- `verificationEmailValidity(email)`
- `addPermission(tournamentName, email)`
- `deletePermission(tournamentName, email)`
- `verificationBattleValidity(battleName, details)`
- `addBattle(battleName, details)`
- `getAllTournamentsAvailable()`
- `subscribeStudent(u_id, tournamentName)`
- `subscribeToBattle(u_id, battleName)`
- `verificationEmailsValidity(emailList)`
- `getMessages(u_id)`
- `notify(u_id, answer, notification_id)`
- `closeTournament(tournamentName)`
- `getBattles(tournamentName)`
- `registerEvaluation(solution)`

Dispatcher

- `verificationRequest(email)`
- `createNewStudent(email, pw, personalInfo)`
- `createNewEducator(email, pw, personalInfo, organization)`
- `sendCredentials(email, pw)`
- `getTournaments(u_id)`
- `verificationNameValidity(tournamentName)`
- `addTournament(tournamentName, details)`
- `verificationEmailValidity(email)`
- `addPermission(tournamentName, email)`
- `deletePermission(tournamentName, email)`
- `verificationBattleValidity(battleName, details)`
- `addBattle(battleName, details)`
- `getAllTournamentsAvailable()`

- `subscribeStudent(u_id, tournamentName)`
- `subscribeToBattle(u_id, battleName)`
- `verificationEmailsValidity(emailList)`
- `getMessages(u_id)`
- `notify(u_id, answer, notification_id)`
- `declineAllInvitations(u_id, notification_id)`
- `closeTournament(tournamentName)`
- `getBattles(tournamentName)`
- `registerEvaluation(solution)`

TournamentManager

- `getTournaments(u_id)`
- `verificationNameValidity(tournamentName)`
- `addTournament(tournamentName, details)`
- `verificationEmailValidity(email)`
- `addPermission(tournamentName, email)`
- `deletePermission(tournamentName, email)`
- `verificationBattleValidity(battleName, details)`
- `addBattle(battleName, details)`
- `getAllTournamentsAvailable()`
- `subscribeStudent(u_id, tournamentName)`
- `verificationEmailsValidity(emailList)`
- `closeTournament(tournamentName)`

BattleManager

- `subscribeToBattle(u_id, battleName)`
- `checkAcceptedInvitations(n_acceptedInvitations, n_requiredInvitations)`

TeamManager

- `subscribeToBattle(u_id, battleName)`

EvaluationManager

- `getBattles(tournamentName)`
- `registerEvaluation(solution)`

RegistrationManager

- `verificationRequest(email)`
- `createNewStudent(email, pw, personalInfo)`
- `createNewEducator(email, pw, personalInfo, organization)`

LoginManager

- `sendCredentials(email, pw)`

NotificationManager

- `verificationRequest(email)`
- `addingTournament(tournamentName, details)`
- `addPermission(tournamentName, email)`
- `deletePermission(tournamentName, email)`
- `addBattle(battleName, details)`
- `getMessages(u_id)`
- `notify(u_id, answer, notification_id)`
- `declineAllInvitations(u_id, notification_id)`
- `closeTournament(tournamentName)`

GitHubManager

- `addBattle(battleName, details)`
- `forward(solution, battle_id)`

CodacyManager

- `forward(solution, battle_id)`

Model

- `createNewStudent(email, pw, personalInfo)`
- `createNewEducator(email, pw, personalInfo, organization)`
- `checkCredentials(email, pw)`
- `getTournaments(u_id)`
- `checkNameValidity(tournamentName)`
- `addTournament(tournamentName, details)`
- `checkEmailValidity(email)`
- `addPermission(tournamentName, email)`
- `deletePermission(tournamentName, email)`
- `checkBattleValidity(battleName, details)`
- `addBattle(battleName, details)`
- `getAllTournamentsAvailable()`
- `subscribeStudent(u_id, tournamentName)`
- `subscribeToBattle(u_id, battleName)`
- `checkEmailsValidity(emailList)`
- `getMessages(u_id)`
- `notify(u_id, answer, notification_id)`
- `declineAllInvitations(u_id, notification_id)`
- `updateScore(team_id, battle_id, score)`
- `closeTournament(tournamentName)`
- `getBattles(tournamentName)`
- `registerEvaluation(solution)`

CKB_DBMS

- `createNewStudent(email, pw, personalInfo)`
- `createNewEducator(email, pw, personalInfo, organization)`
- `checkCredentials(email, pw)`
- `getTournaments(u_id)`
- `checkNameValidity(tournamentName)`
- `addTournament(tournamentName, details)`
- `checkEmailValidity(email)`
- `addPermission(tournamentName, email)`
- `deletePermission(tournamentName, email)`
- `checkBattleValidity(battleName, details)`
- `addBattle(battleName, details)`
- `getAllTournamentsAvailable()`
- `subscribeStudent(u_id, tournamentName)`
- `subscribeToBattle(u_id, battleName)`
- `checkEmailsValidity(emailList)`
- `getMessages(u_id)`
- `notify(u_id, answer, notification_id)`
- `declineAllInvitations(u_id, notification_id)`
- `updateScore(team_id, battle_id, score)`
- `closeTournament(tournamentName)`
- `getBattles(tournamentName)`
- `registerEvaluation(solution)`

email Provider

- `sendEmail(email, text)`
- `sendEmail(emailList, text)`

GitHub

- `createRepository(battleName, details)`

Codacy

- `checkSolution(solution)`
- `forward(solution, battle_id)`

2.6 Selected architectural style and patterns

2.6.1 Client-Server Architecture

The CKB system uses the Client-Server architecture to provide its services to the clients. In this kind of architecture, the "actors" of the communication process are divided into two categories: clients and servers. Clients are the active machines that request the use of the services made available by the server, in this case, clients are the devices used by the users to access the CKB system. On the other hand, servers are the passive components, those who answer the requests made by the clients, in this case, the passive components are the servers used by the CKB platform. There is of course more than one server because the system aims to provide a good degree of availability and reliability, characteristics that require redundancy in the architecture.

2.6.2 Three-Tiered Architecture

The CKB client-server architecture will be organized into three tiers, each being responsible for a specific part of the communication process. The first one is the data tier and it's represented by the database on which the CKB system relies to store information about users, battles, etc. The data tier is supposed to be accessed only by the Application Server, in particular by the Model described in section 2.2. The second tier is the application tier, this tier corresponds to the Application Server and contains the business logic of the system. The main task of the application tier is to provide the function of the CKB system to its users through appropriate interfaces (described in section 2.5) and to communicate with both the data tier, through the Model, and the presentation tier, with the use of the Web Server. As already stated, the last layer of the CKB system is the presentation tier, which is divided between the Web Server and the clients. The main goal of the presentation tier is to provide a user-friendly interface to the users that facilitates them in the use of the services provided by the application tier, the main parts that compose the user interface are divided into static (HTTP pages) and dynamic (functions that require the interaction with the Web Server and the application tier). Overall this kind of model provides a good degree of flexibility thanks to the division of the different functions of the system into ad-hoc layers that can be easily modified and maintained without impacting the others.

2.6.3 Model-View-Controller

The CKB system is implemented using the Model-View-Controller design pattern. The use of this pattern allows the system to divide its functions into three parts, which then can be modified and maintained more efficiently and without conflicts. The MVC pattern separates the system into three parts: The Model (also represented in section 2.2) is responsible for handling the representation of data in the system and is never to be accessed directly by the users. The Controller contains all the functions that the system offers and through which the users can access and modify the model. The Controller is divided into different parts (the "managers"), which are each one responsible for the computations about a specific part of the system functions (e.g. functions regarding tournaments, login, etc.). The last part is the View, which is responsible for the interface given to the users to access the functionalities of the CKB system, the View also handles the connection between the inputs coming from the interaction with the UI and the Controller, in other words, the user employs the interface to interact with the Controller and request its services.

2.6.4 Circuit Breaker

The Circuit Breaker architectural pattern is implemented for all the remote services made available by the CKB system, this pattern monitors each service call that comes from the clients and makes them fail if they incur too many errors, this is important to avoid ripple effects caused by malfunctioning processes and clients don't risk to call one of those malfunctioning services. The circuit breaker pattern is also used to monitor access and availability of the DBMS.

2.7 Other design decisions

2.7.1 Remote Method Invocation

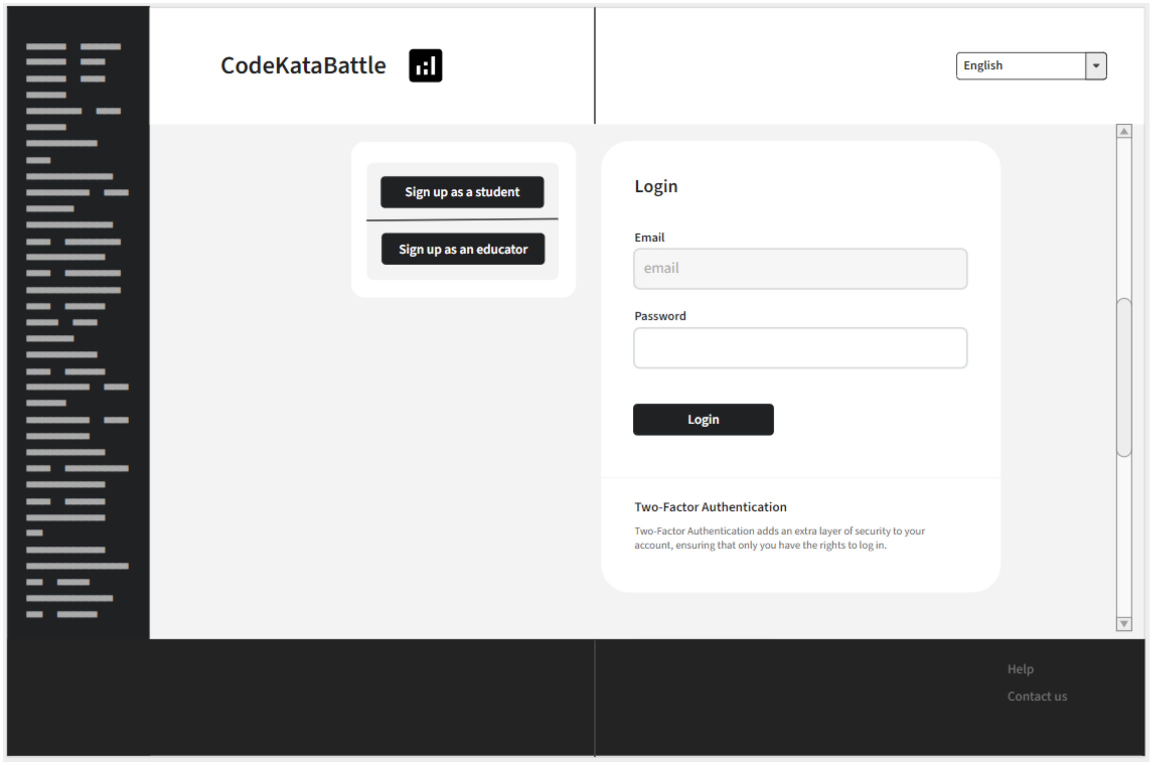
The communication between the users and the CKB system is handled using the Remote Method Invocation protocol (RMI), this kind of protocol requires each component to provide a proxy with an interface with methods that external agents can use to request the execution of the services located on the components. RMI provides a good degree of homogeneity in the code of the application since both local and remote method invocations are performed in the same fashion and are well integrated with an object-oriented environment.

2.7.2 Server Replication

The servers operating for the CKB system and for the DBMS are replicated, resulting in a set of synchronized machines able to divide the workload coming from the clients and the third-party systems. Replication does not only concern the workload, but it also aims to achieve fault tolerance, so that even if a single machine (or a small group of them) fails, the other ones can compensate until the failed machines recover.

3 User interface design

This section provides an expanded set of sample parts of the user interface.



Two-Factor Authentication adds an extra layer of security to your account, ensuring that only you have the rights to log in.

Figure 22: User Interface: Login form

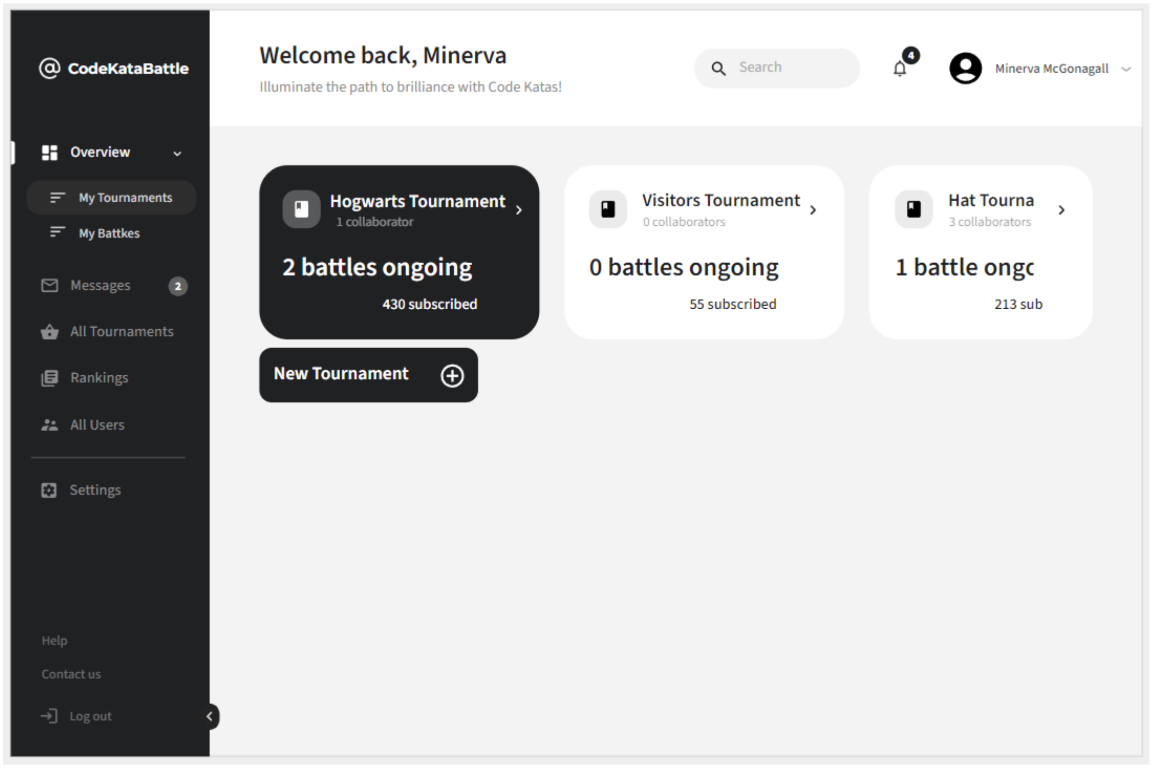
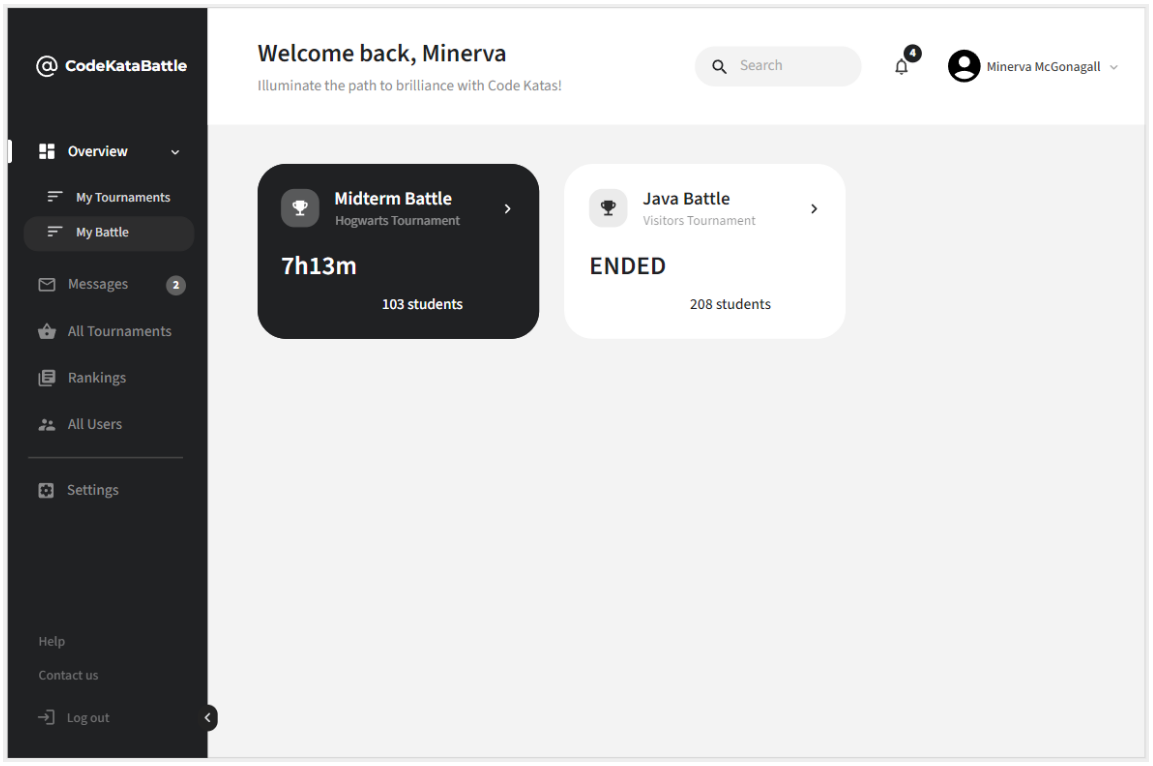


Figure 23: User Interface: Educator's MyTournaments Tab



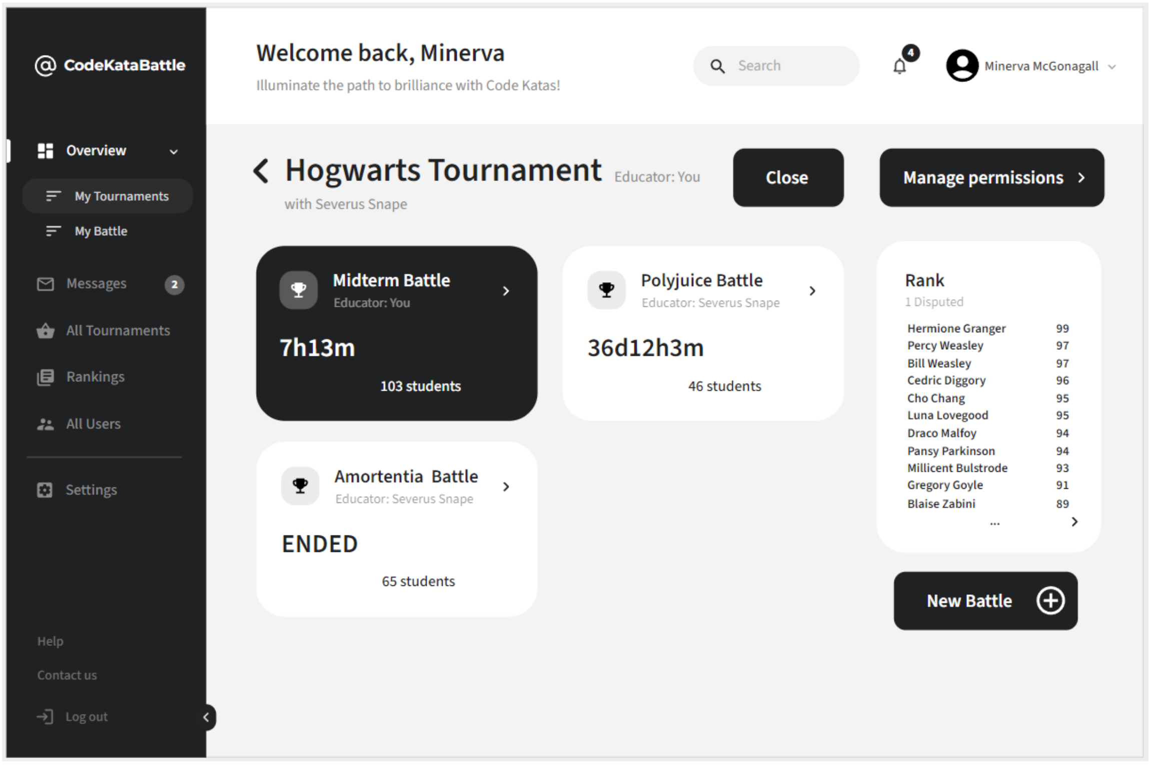
Welcome back, Minerva

Illuminate the path to brilliance with Code Katas!

Midterm Battle
Hogwarts Tournament
7h13m
103 students

Java Battle
Visitors Tournament
ENDED
208 students

Figure 24: User Interface: Educator's MyBattles Tab



Welcome back, Minerva

Illuminate the path to brilliance with Code Katas!

Hogwarts Tournament Educator: You

with Severus Snape

Midterm Battle
Educator: You
7h13m
103 students

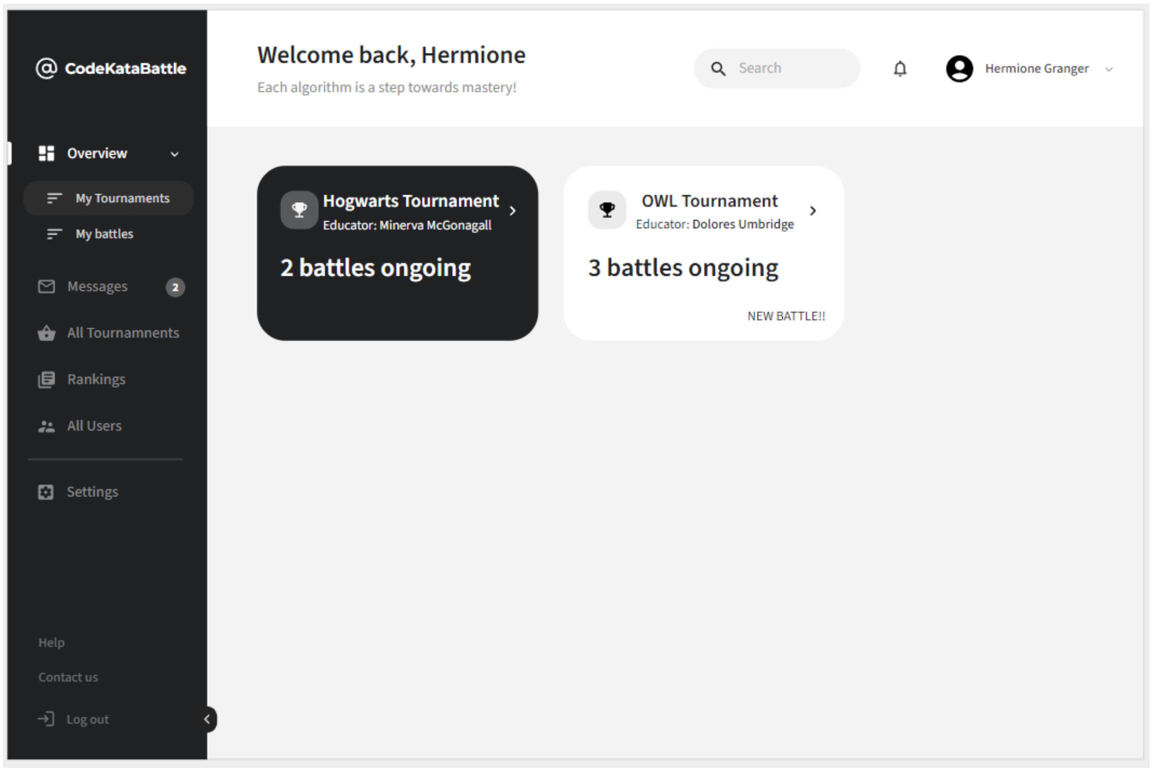
Polyjuice Battle
Educator: Severus Snape
36d12h3m
46 students

Amortentia Battle
Educator: Severus Snape
ENDED
65 students

Rank
1 Disputed

Hermione Granger	99
Percy Weasley	97
Bill Weasley	97
Cedric Diggory	96
Cho Chang	95
Luna Lovegood	95
Draco Malfoy	94
Pansy Parkinson	94
Millicent Bulstrode	93
Gregory Goyle	91
Blaise Zabini	89
...	

Figure 25: User Interface: Educator's Tournament Information Tab



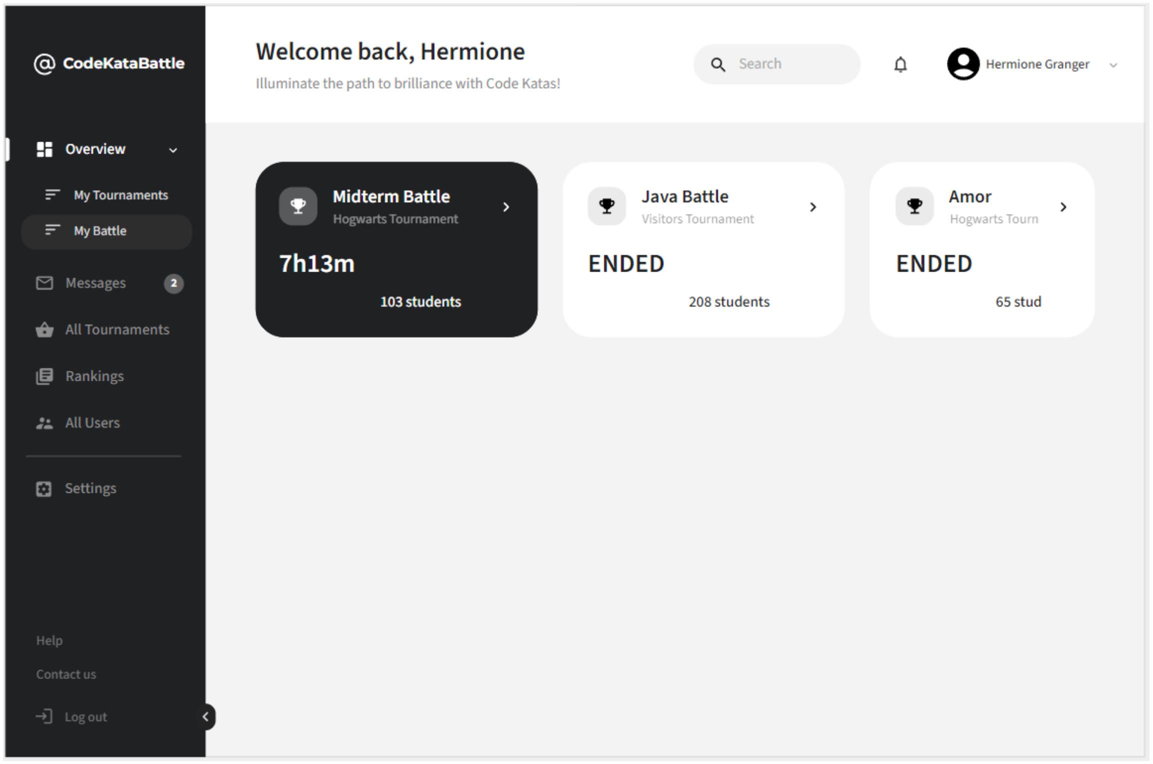
Welcome back, Hermione

Each algorithm is a step towards mastery!

Hogwarts Tournament
Educator: Minerva McGonagall
2 battles ongoing

OWL Tournament
Educator: Dolores Umbridge
3 battles ongoing
NEW BATTLE!!

Figure 26: User Interface: Student's MyTournaments Tab



Welcome back, Hermione

Illuminate the path to brilliance with Code Katas!

Midterm Battle
Hogwarts Tournament
7h13m
103 students

Java Battle
Visitors Tournament
ENDED
208 students

Amor
Hogwarts Tourn
ENDED
65 stud

Figure 27: User Interface: Student's MyBattles Tab

4 Requirements traceability

This section describes the relation between the requirements defined in the RASD and the architectural component of section 2.2. In particular each requirement will have a set of components assigned to it that represents the necessary elements of the architecture needed to provide that requirement.

Requirements	[**R1.**] The system allows allow all users to log in
Components	• Web App • Web Server • Dispatcher • LoginManager • Model • CKB DBMS

Requirements	[**R2.**] The system allows users to register with the proper account
Components	• Web App • Web Server • Dispatcher • RegistrationManager • NotificationManager • Model • email Provider • CKB DBMS

Requirements	[**R3.**] The system allows educators to create tournaments [**R5.**] The system allows educators with the proper permissions to create a battle for a tournament [**R7.**] The system allows students to register to a tournament [**R21.**] The system allows educators to close tournaments they created [**R23.**] The system allows users to consult the ranking of a tournament
Components	• Web App • Web Server • Dispatcher • TournamentManager • Model • CKB DBMS

Requirements	[**R4.**] The system allows educators to grant permissions to manage a tournament to other educators, but not to themselves
Components	• Web App • Web Server • Dispatcher • TournamentManager • NotificationManager • Model • email Provider • CKB DBMS

Requirements	[**R6.**] The system allows educators to insert all the information regarding a battle during the battle creation phase [**R22.**] The system allows users involved in a battle to consult the partial score of that battle
Components	• Web App • Web Server • Dispatcher • TournamentManager • BattleManager • Model • CKB DBMS

Requirements	[**R8.**] The system notifies students when a new is created [**R18.**] The system notifies students registered to a tournament when a new battle belonging to that tournament is created [**R20.**] The system notifies students when a tournament they are registered to ends
Components	• TournamentManager • NotificationManager • Model • email Provider • CKB DBMS

Requirements	[**R9.**] The system allows students to register for a battle
Components	• Web App • Web Server • Dispatcher • TournamentManager • BattleManager • TeamManager • Model • CKB DBMS

Requirements	[**R10.**] The system doesn't allow a student already registered in a battle with a team to be registered in the same battle with another team
Components	• BattleManager • TeamManager • Model • CKB DBMS

Requirements	[**R11.**] The system allows students to send invitation to other students but not to themselves
Components	• Web App • Web Server • Dispatcher • BattleManager • TeamManager • NotificationManager • Model • email Provider • CKB DBMS

Requirements	[**R12.**] The system allows invited students to accept or decline an invitation
Components	• Web App • Web Server • Dispatcher • TeamManager • NotificationManager • Model • email Provider • CKB DBMS

Requirements	[**R13.**] The system creates a GitHub repository for each battle [**R14.**] The system allows GitHub to create an automated workflow connected to the CKB platform [**R15**.] The system allows GitHub to send submitted solutions to the CKB platform
Components	• GitHubManager • Model • GitHub • CKB DBMS

Requirements	[**R16.**] The system automatically evaluates every proposed solution with reference to the specified parameters
Components	• GitHubManager • CodacyManager • Model • GitHub • Codacy • CKB DBMS

Requirements	[**R17.**] The system allows educators to manually evaluate solutions at the end of the battle, if specified
Components	• Web App • Web Server • Dispatcher • BattleManager • EvaluationManager • Model • CKB DBMS

Requirements	[**R19.**] The system notifies students when a battle they are registered to ends
Components	• BattleManager • NotificationManager • Model • email Provider • CKB DBMS

Requirements	[**R24.**] The system updates the ranking of a tournament at the end of each battle belonging to that tournament [**R25.**] The system doesn't allow a tournament to be closed if a battle is still ongoing [**R29.**]The system doesn't allow students to subscribe to tournaments or battle after the registration deadline
Components	• BattleManager • TournamentManager • Model • CKB DBMS

Requirements	[**R26.**] The system doesn't allow to create a tournament with the same name of an already existing one.
Components	• TournamentManager • Model • CKB DBMS

Requirements	[**R27.**] The system automatically declines unanswered invitations that are older than 2 days [**R28.**] When a student answers an invitation the system automatically declines all the other invitations sent to that student
Components	• TeamManager • NotificationManager • Model • email Provider • CKB DBMS

Requirements	[**R30.**]The system cannot receive submitted solutions after the deadline
Components	• BattleManager • GitHubManager • Model • GitHub • CKB DBMS

5 Implementation, integration and test plan

5.1 Overview

This section presents to the reader the implementation and testing processes of the CKB system Application Server. The implementation part will be separated into modules, each of them adding a specific class of functionalities, that will be implemented gradually, in this way, testing can start almost as soon as the implementation of the system. The goal of testing will be to test the limits of a certain implementation and understand what makes it fail, to provide better coverage, to achieve this result the modules will not only be tested as a single entity but also in the way they interact with the other components of the system. The integration testing of the CKB system will be carried out using a bottom-up approach, starting from the central parts of the system and gradually adding and testing more functionalities, and will include both static and dynamic testing. It is also paramount that the code is documented during all the phases of the implementation process.

5.2 Milestones

The implementation of the CKB system will be articulated into a series of milestones that follow the bottom-up approach, these milestones will represent the main functionalities of the system, that will gradually be implemented and tested.
Precisely, the following milestones have been identified:

- M1. Model
- M2. Registration and Login features
- M3. Tournament Management features
- M4. Battle Management features
- M5. Integration with third-party components
- M6. Notification features
- M7. Dispatcher

Each of these modules will also be paired with a set of drivers that will simulate the use of the implemented components.

M1. The first milestone is the Model implementation, this is the first implemented component because every other module will rely on the representation of data contained in the model, for this specific reason it will have to be thoroughly tested since a malfunction in the Model which is not discovered in this phase could negatively impact the functioning of the successive modules.

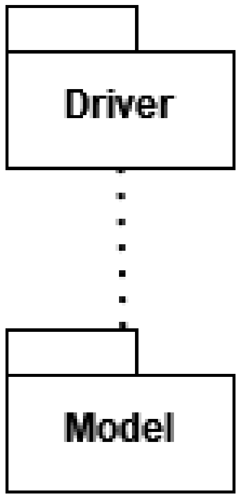


Figure 28: First milestone

M2. The second milestone consists of the registration and login features, incarnated by the RegistrationManager and the LoginManager. These components will be integrated with the Model in order to test the stability of these kinds of processes and the resistance against malicious actors, such as processes of SQL injection coming from the registration or login forms.

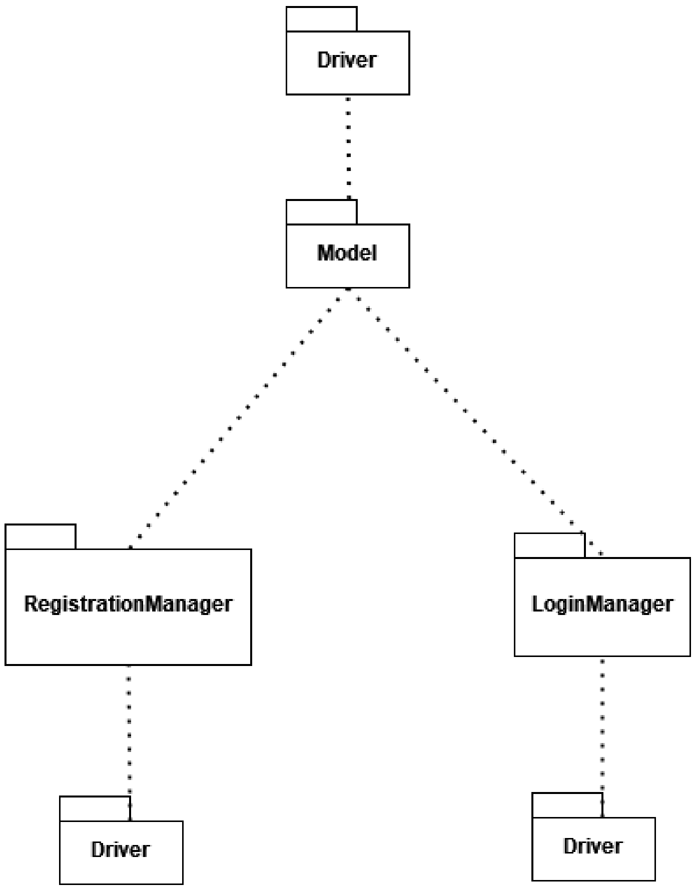


Figure 29: Second milestone

M3. The third part of the integration process is the development of the TournamentManager, which handles the functionalities regarding tournaments on the CKB platform. The testing on this part will also focus on the integration between the TournamentManager and the users created in the milestone 5.2, to check if the users are created with the correct permissions and can correctly interact with the tournament objects.

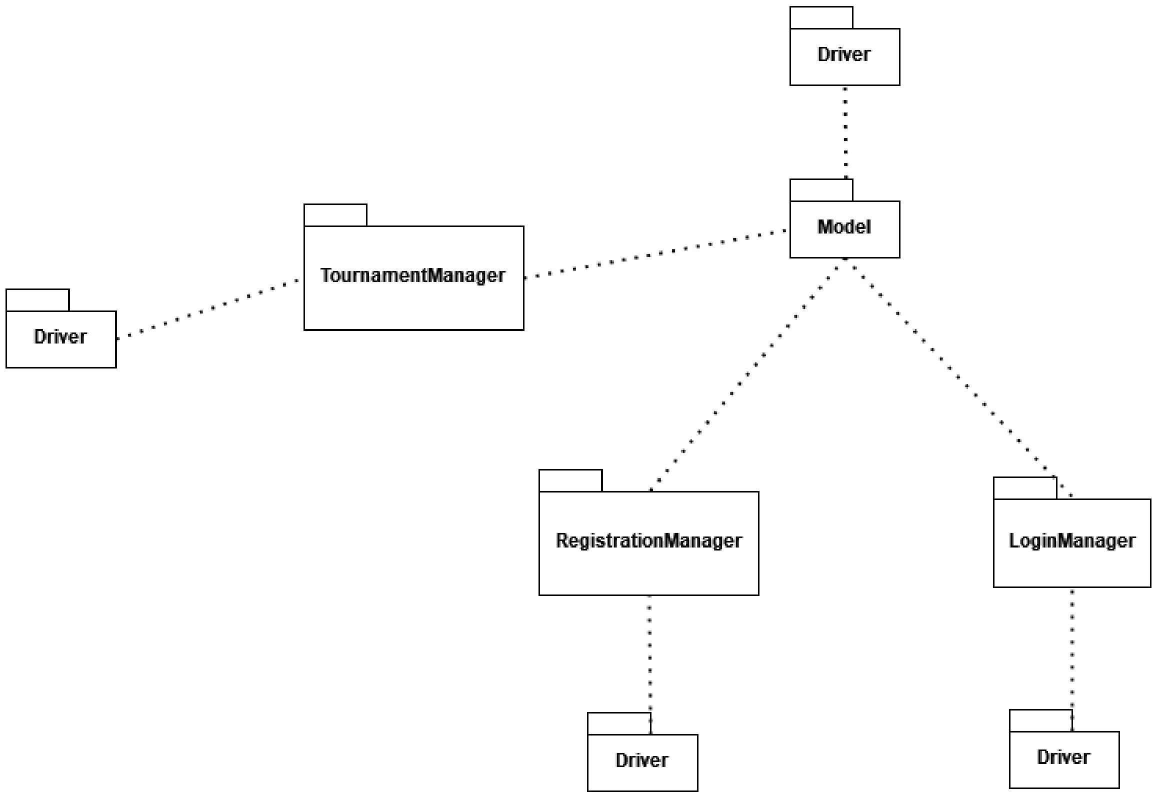


Figure 30: Third milestone

M4. After implementing the Tournament features the focus will be directed towards the implementation of the battle management capabilities, which will include not only the BattleManager, responsible for providing the interface to manage battles, but also the TeamManager, which manages the process of subscription of a team to a battle. Since these two components are strongly connected it's advantageous to implement them in parallel. Once the BattleManager and the TeamManager are implemented and tested on their own the focus of the testing process will be directed towards the integration testing with the TournamentManager, to test all of the tournament process including battle creation and ranking updates.

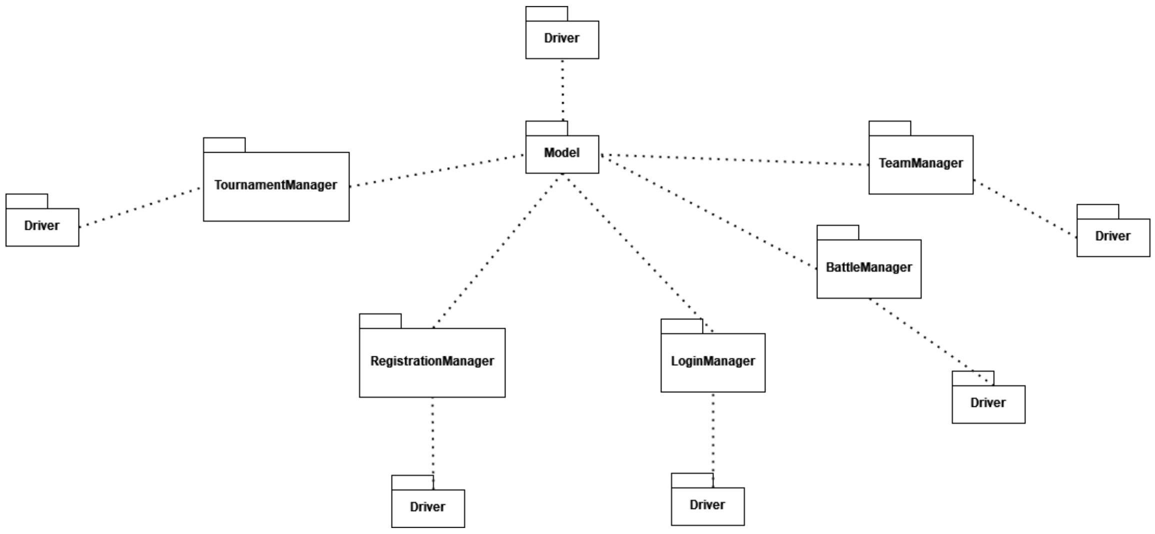


Figure 31: Fourth milestone

M5. The fifth milestone focuses on the implementation of the managers that interact with the third party components (GitHub and Codacy) and on the interaction between them. The testing phase will firstly take into account the interaction between the newly added components, and then the interaction with the BattleManager, since battles can be tested completely only with the contribution of these components.

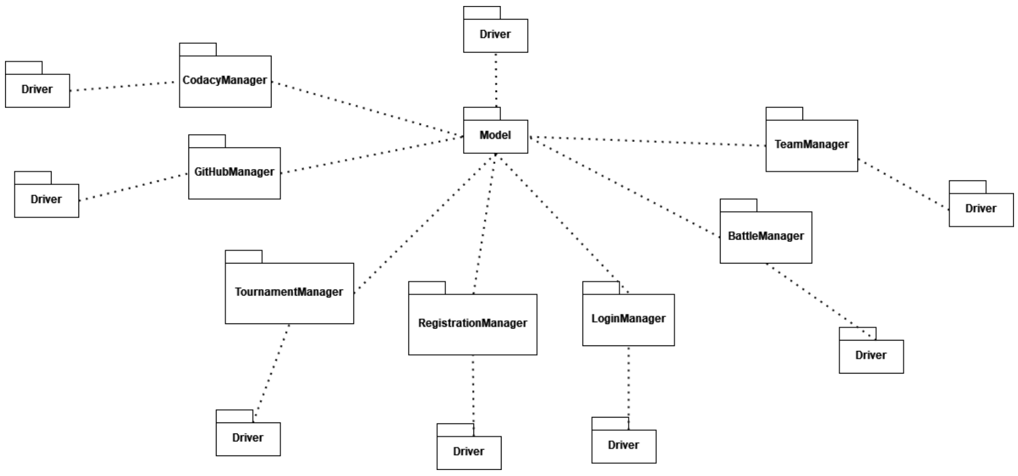


Figure 32: Fifth milestone

M6. The sixth milestone regards the implementation of the notification system, since at this point the core functions of CKB should have already been implemented and, at least in part, tested. The NotificationManager is implemented this late because it interacts with a great amount of components, and would not have been able to be tested before the implementation of those functionalities. Testing of the NotificationManager will involve every other part already implemented into the system and will check the correct functioning of every kind of notification to every category of user.

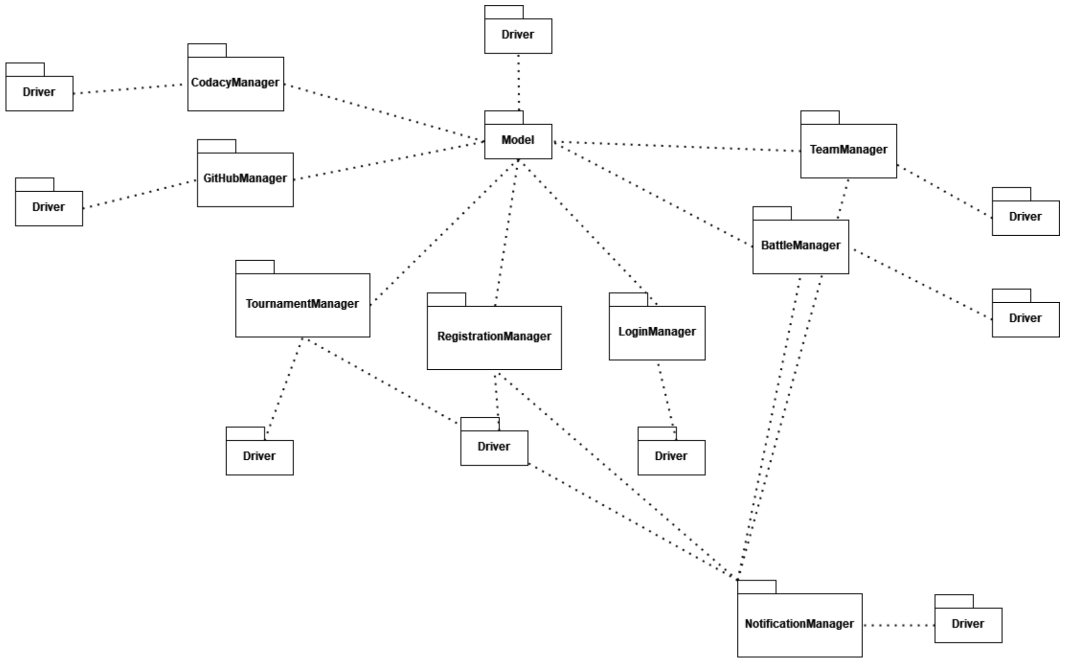


Figure 33: Sixth milestone

M7. The seventh and final milestone focuses on the implementation of the components that will interact with the external environment, first among them the Dispatcher, whose task is to redirect to the correct part of the controller the incoming messages. This phase is also highly interconnected with the development of the Web Server. The testing of the Dispatcher also involves almost every of the components previously implemented, for obvious reasons and will require the simulation of the requests coming from the Web Server.

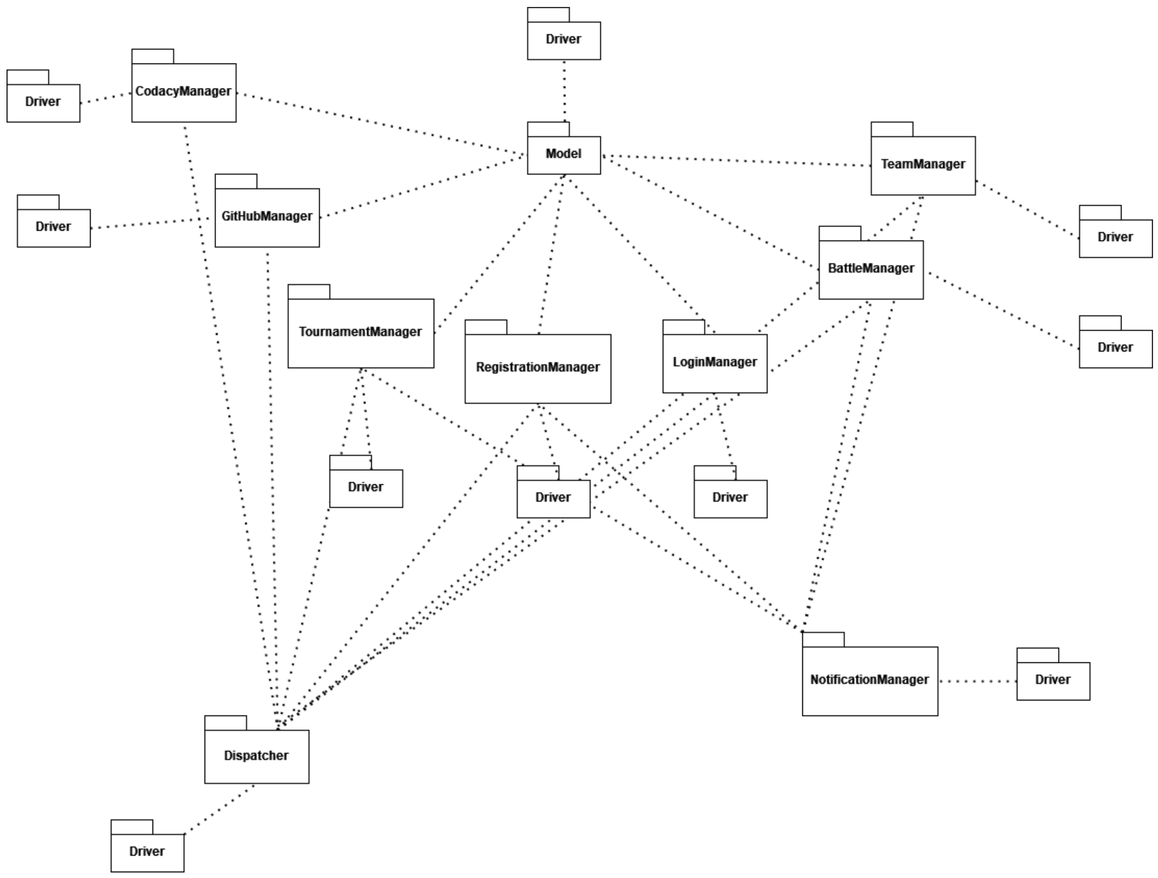


Figure 34: Seventh milestone

5.3 System Testing

After every component of the CKB system has been implemented, the focus of the testing effort will shift to testing the system as a whole, which, in advanced stages, could also involve the contribution of a selected group of external people that would act as beta testers, to simulate the use cases from the perspective of a group of people that were not involved in the development of the system and to verify the effectiveness of the user interface. After the functionality tests have been conducted successfully it will also be necessary to test the general performance of the CKB system and how the system behaves in critical situations, such as a high number of requests coming from different clients. To this end the system will undergo a series of performance tests, aimed at measuring the main performance metrics (such as the answer time from different parts of the Application Server), followed by a phase of stress testing, that will try to understand the weaknesses of the CKB system in case of a high number of requests, these requests could come from a single ”source” (e.g. a lot of traffic coming from clients, or several requests by GitHub at the same time) or from different sources.

5.4 Future Development

The CKB system will of course need to be updated also after the release of the first version to the public, not only to correct bugs that may not have been found in the testing phase but also to add new features to the CKB platform. This operation will require a road map similar to the one described in the previous sections but on a smaller scale. The new features will be tested on their own during the implementation phase and then will be integrated with the preexisting components, then the system will be tested as a whole by the development team and released as a new beta version, tested by a selected group of beta testers and then fully released to the public.

6 Effort spent

6.1 Francesco Galbiati

Chapter 1:	0h00m
Chapter 2:	12h30m
Chapter 3:	0h00m
Chapter 4:	2h30m
Chapter 5:	3h00m

6.2 Chiara Fossà

Chapter 1:	4h
Chapter 2:	26h15m
Chapter 3:	1h15m
Chapter 4:	0h45m
Chapter 5:	0h15m

7 References

Document structure: Project assignment document and previous year project (https://webeep.polimi.it/mod/folder/view.php?id=219353)
UI mockups: Moqups (https://moqups.com/it/templates/wireframes-mockups/)
Diagrams and schemes: Draw.io(http://draw.io/)
Lecture material from the Software Engineering 2 course at Politecnico di Milano