Design Document

Politecnico di Milano
Merlino Lorenzo & Iodice Andrea

January 1, 2024

Contents

1 Introduction

1.1 Purpose

The System to Be will allow users to take part in coding battles. Each coding battle will be part of a tournament and consists of a coding project to be completed. Educators will create tournaments and create battles by supplying the project structure. Students will be able to create teams and the teams will be able to submit a solution to the coding battle. Each solution will be scored using different criterias, such as the amount of test cases passed, amount of time required to submit the solution and quality of the code provided. Some criteria will be automatically asserted and the remaining ones will be asserted by the creator of the battle.

1.2 Scope

CodeKataBattle (CKB) is a new platform that helps students improve their software development skills by training with peers on code kata.

Educators use the platform to challenge students by creating code kata battles in which teams of students can compete against each other, thus proving and improving their skills.

Educators create tournaments to which students can enroll. After enrolling in a tournament students can take part in code kata battle by joining or creating a team.

A code kata battle is a programming exercise. The exercise includes a brief textual description and a software project with build automation scripts that contains a set of test cases that the program must pass.

Teams participating in a battle are expected to follow a test-first approach and develop a solution that passes the required tests. Groups deliver their solution to the platform before the deadline. At the end of the battle, the platform and the educators assign scores to groups, in order to create a competition rank.

A Educator cannot be a student and vice versa.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Coding Battle**: programming exercise proposed by an Educator in the context of a tournament.
- **Tournament**: a series of coding battles, in which Students can take part as teams. It's created and managed by an Educator.
- **Team**: group of students who enroll in a tournament. Students in the same team work together and the solution they submit will be taken into consideration for each member of the team.
- **Rank**: position of a Student in a tournament's leaderboard, which is calculated on each Student's score.
- **Score**: points gained by a Student by submitting a solution to a Coding Battle.
- **Open battle**: a battle that accepts new submissions.
- **Closed battle**: a battle that doesn't accept new submissions but some still need to be evaluated.
- **Completed battle**: a battle that doesn't accept new submissions and where all the submissions have been evaluated.
- **Open tournament**: A tournament where new battles can be created.

- **Closed tournament**: A tournament where new battles cannot be created but some are still not completed.
- **Completed tournament**: A tournament where new battles cannot be created and all battle have been completed
- **Team leader**: Student who created the team.
- **Automated grading**: A series of scores that is automatically assigned to a solution, it is composed of:
  - Number of test passed
  - Time passed after the battle publication
  - Static analysis result
- **User**: either a Student or an Educator
- **Educator**: user of the system, who manages tournaments and battles.
- **Student**: user of the system, who participates in tournaments and battles.

1.3.2 Acronyms

- **RASD**: Requirements Analysis and Specification Document
- **DD**: Design Document

1.3.3 Abbreviations

- [**Rn**]: the n-th requirement
- [**UCn**]: the n-th use case

1.4 Revision History

- Version 1.0

1.5 Reference Documents

- The specification of the RASD and DD assignment of the Software Engineering II course, held by professor Matteo Rossi, Elisabetta Di Nitto and Matteo Camilli at the Politecnico di Milano, A.Y 2023/2024;
- Slides of Software Engineering 2 course on WeBeep;
- DD Sample from A.Y. 2022-2023
- Github API documentation

1.6 Document Structure

Mainly the current document is divided in 7 chapters, which are:

1. Introduction: The first chapter includes the introduction in which is explained the purpose of the document, then, a brief recall of the concepts introduced in the RASD is given. Finally, important information for the reader is given, i.e. definitions, acronyms, synonyms and the set of documents referenced.

2. Architectural Design: it includes a detailed description of the architecture of the system, including the high level view of the elements, the software components of CKB, a description through runtime diagrams of various functionalities of the system and, finally, an in-depth explanation of the architectural pattern used.

3. User Interface Design: are provided the mockups of the application user interfaces, with the links between them to help in understanding the flow between them.

4. Requirements Traceability: it describes the connections between the requirements defined in the RASD and the components described in the first chapter. This is used as proof that the design decisions have been taken with respect to the requirements, and therefore that the designed system can fulfill the goals.

5. Implementation,Integration and Test Plan: it describes the process of implementation, integration and testing to which developers have to stick in order to produce the correct system in a correct way.

6. Effort spent

7. References: it contains the references to any documents and to the Software used in this document.

2 Architectural design

2.1 Overview

This section describes the overview of the system from a design standpoint, defining its main components and their relationships

2.1.1 High level view

The CodeKata system is a distributed system with a four-tier architecture. The four tiers are:

- Presentation: displays the user interface and manages the user interactions.
- Web Server: manages the information sent by the presentation layer, is in charge of load balancing and caching.
- Application: handles the system logic managing authentication and requests from the presentation layer. It filters access to the data layer
- Data: maintains all persistent data of the system and the shared data between different instances of the application layer

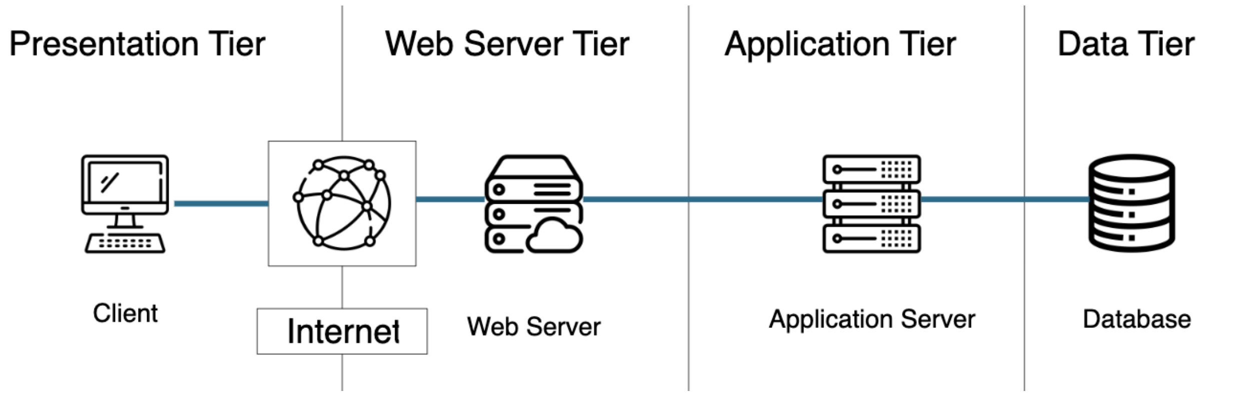


Figure 1: High level view

The communication between client and server through http requests and responses. The system can be accessed via browser by both Students and Educators.

The presentation tier communicates with the web server layer which is in charge of distributing the request between the multiple application layer instances.

The application tier elaborates the requests and access the data from the data layer. The application tier interfaces with the GitHub api on the user behalf.

User session is identified a by JWT, Json Web Token. it contains the user internal ID, the user Github username and the user Github access token

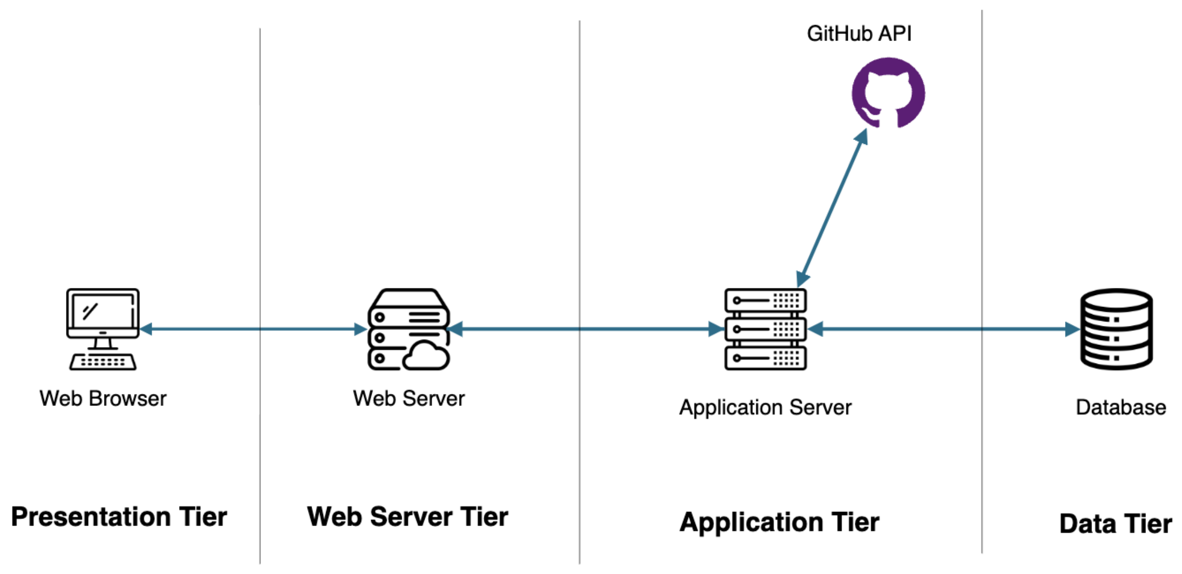


Figure 2: High level view

2.1.2 Distributed view

The system has multiple instances of the application tier to meet availability and performance requirements.

2.2 Component view

The system is divided into different components, the following diagram describes which components are in the system and how they are connected between each other.

External components that are not part of the system are colored white. Presentation components are colored purple. Web server components are colored green. Application components are colored yellow. Data components are colored blue.

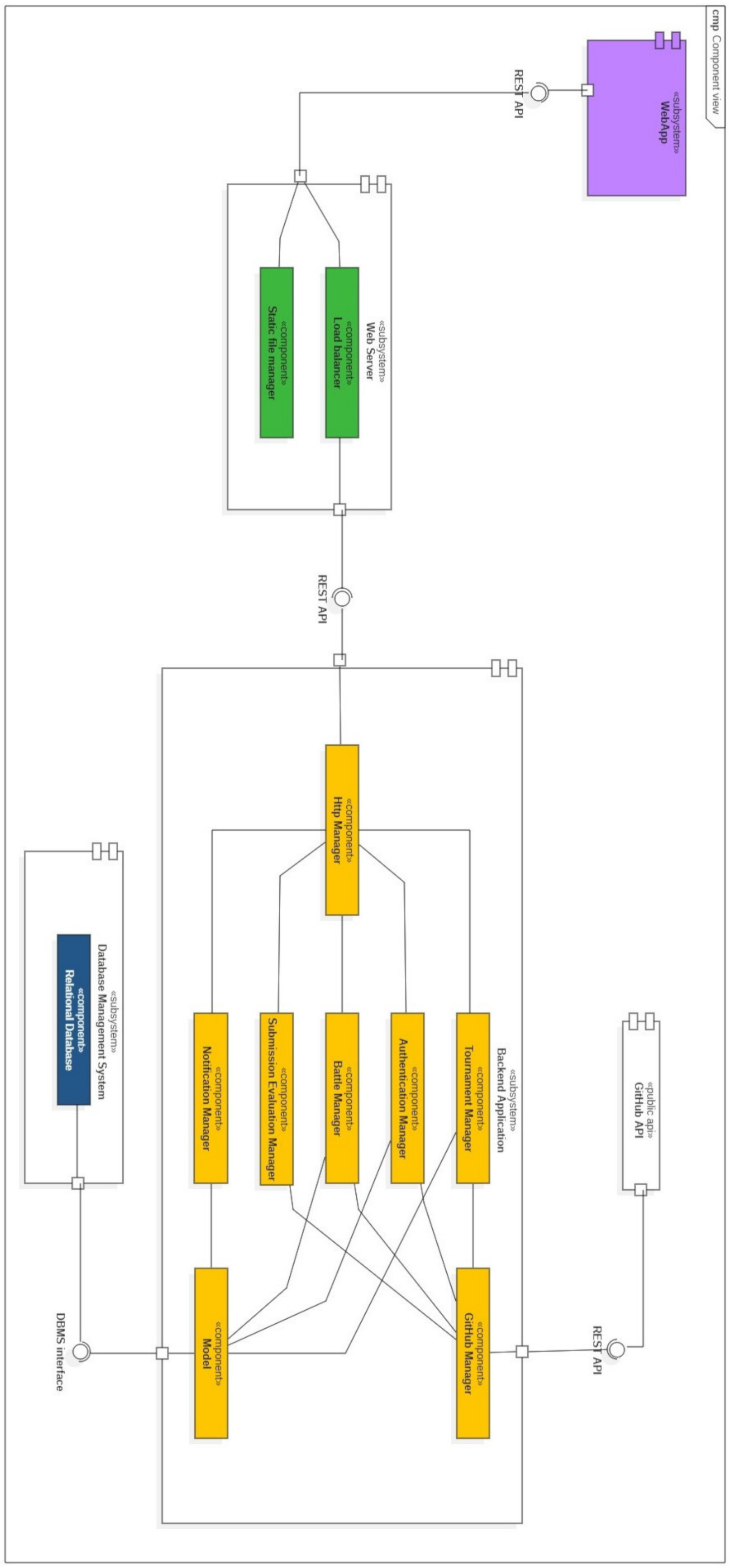

cmp Component view
«subsystem» WebApp
REST API
«subsystem» Web Server
«component» Load balancer
«component» Static file manager
REST API
«subsystem» Backend Application
«component» Http Manager
«component» Tournament Manager
«component» Authentication Manager
«component» Battle Manager
«component» Submission Evaluation Manager
«component» Notification Manager
«component» GitHub Manager
«component» Model
«public api» GitHub API
REST API
«subsystem» Database Management System
«component» Relational Database
DBMS interface

Figure 3: Component view

2.2.1 The Relational Database

contains all the relevant data to the application and it resolves the queries received from the model component with the requested data.

2.2.2 The Model

is used to store the data that are part of the persistent state of the system (which in our case is stored on a database) when this is loaded into the application. Moreover, it provides all the methods that are necessary to access those data and it also implements the logic to manipulate them. In other words: whenever some data contained in the database is needed by any component of the system, for any kind of computation, they delegate the task of retrieving such information to the model component. It not only retrieves them but it also stores them in ad-hoc data structures so that they are ready to be used for the required computations. The model component is the only one interacting with the Relational Database.

2.2.3 The GitHub Manager

manages all interactions between components and the GitHub api.

2.2.4 The Notification Manager

registers the notification's creation and manages its internal status until deletion. It interfaces with the model to record new notifications, retrieve existing ones and delete old ones

2.2.5 The Authentication Manager

handles the user login and interfaces with the GitHub Manager to verify the user access token. It interfaces with the model to record the access token or retrieve user info.

2.2.6 The HTTP Manager

handles http requests and manages the http sessions.

2.2.7 The Battle Manager

handles battles' creation and updates their status when needed. When creating a battle it verifies that the battle configuration is valid, handles teams creations, teams joins and team invites. It interfaces with the Notification manager to notify invites to join a team, when a Student joins a team and when a battle is created or closed. Interfaces with the database to record battles and teams data.

2.2.8 The Tournament Manager

handles the creation of tournaments, the additions of allowed educators into the tournaments, the enrollment of students into tournaments and the status of the tournaments. It interfaces with the database to record and retrieve tournament data.

2.2.9 The Submission Evaluation Manager

handles the automatic grading of new submissions of open battles. Handles manual grading of closed battles. Interfaces with the database to record the team gradings.

2.2.10 The Web-App

is the web interface used by users to access the system and interact with it. It sends requests to the Web Server via HTTP and shows the user the responses from the system.

2.2.11 The Web Server

manages incoming requests from the Web-App component and distributes them between the application's instances. Handles the distributions of static files to the Web-App.

2.3 Deployment view

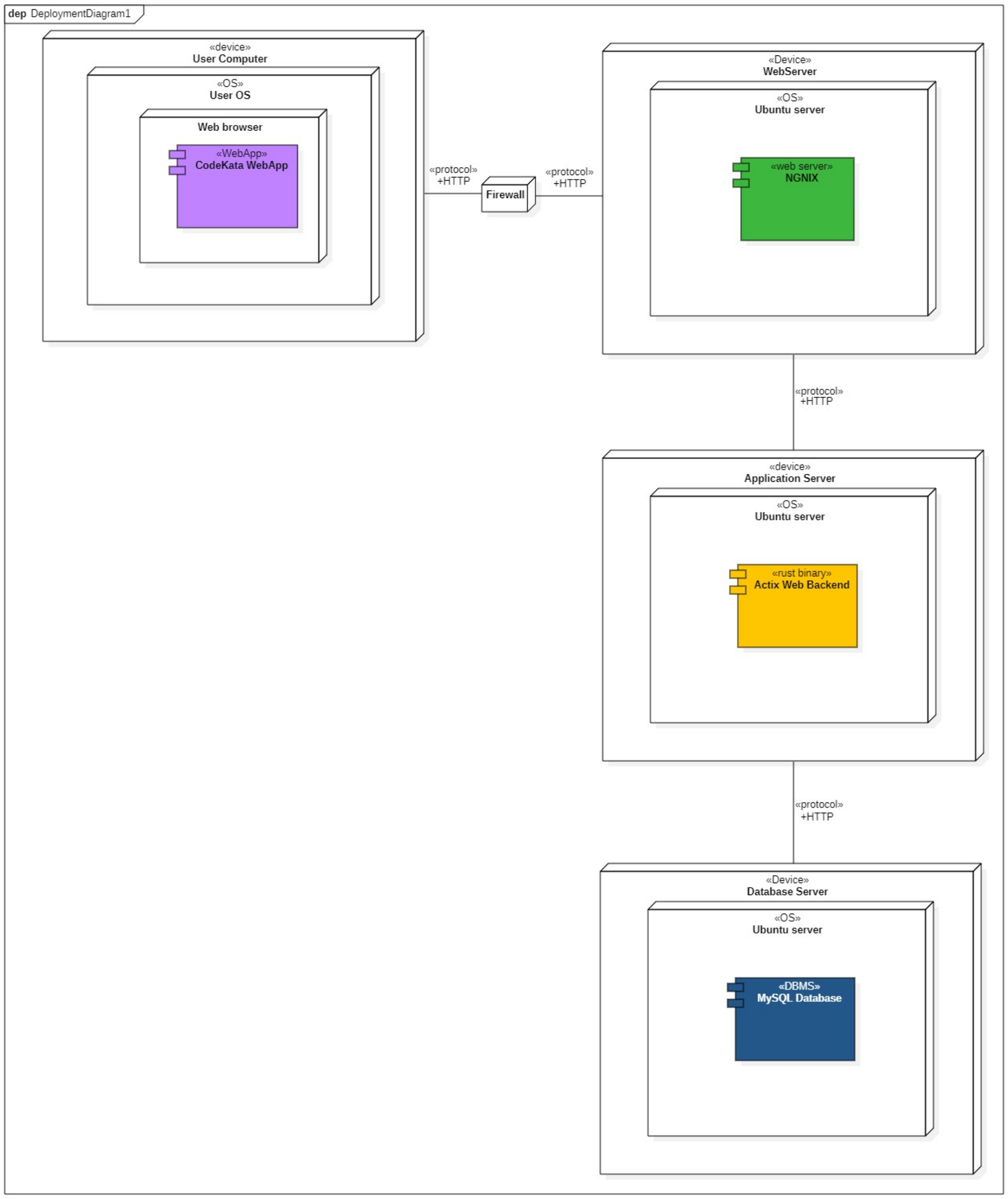


Figure 4: Deployment view

Further details about the elements in the graph are provided in the following.

- **Firewall** is a device that monitors the packets incoming to the system, if a packet is potentially dangerous it is not forwarder. It is placed before the Web Server and allows incoming packers only if directed to the web server with the correct port and the correct protocol, in this way the only packets that enter the network are considered safe.

- **Computer** is a regular computer used by the user. No special applications are required to be installed on it, except for a web browser. This will be necessary to surf the internet and reach the CKB web page. An HTTPS connection with the system is established by passing through a firewall and after the forwarding, operated by the designated load balancer, to an available application server.
- **Web server** receives all the requests sent from the web app. It then forwards the received request to the different instances of the application allowing for the incoming load to be balanced on the different instances . It also distributes the web app static files and handles the caching of requests when needed.
- **Application server** receives all the requests sent from the web server and elaborates them. It sends requests to the Database server to read and write data when needed. It communicates directly with the GitHub api.
- **Database server** host the relational database that contains all the system data it is accessed by the application server.

2.4 Runtime views

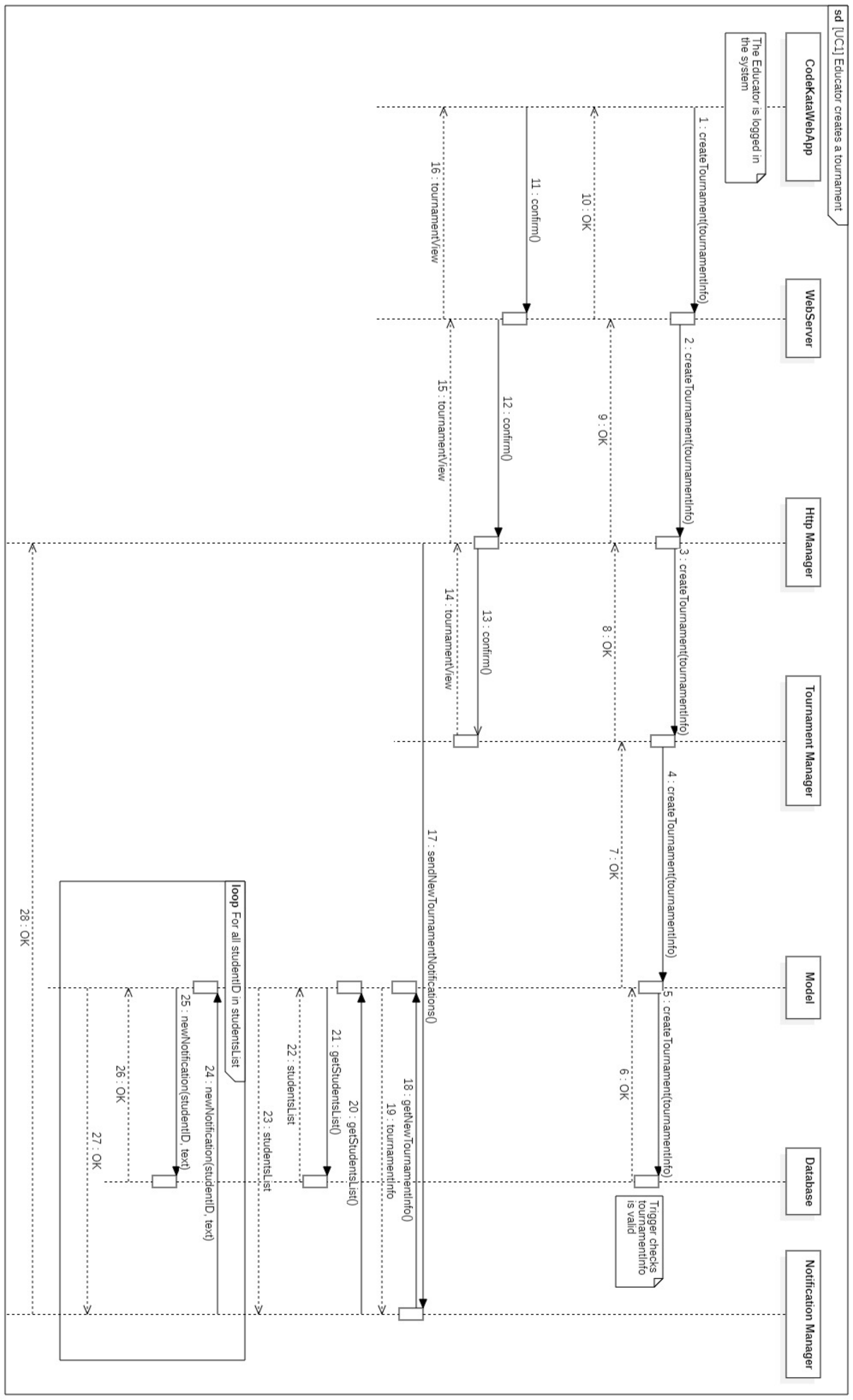

sd [UC1] Educator creates a tournament
CodeKataWebApp
WebServer
Http Manager
Tournament Manager
Model
Database
Notification Manager
The Educator is logged in the system
1 : createTournament(tournamentInfo)
2 : createTournament(tournamentInfo)
3 : createTournament(tournamentInfo)
4 : createTournament(tournamentInfo)
5 : createTournament(tournamentInfo)
Trigger checks tournamentInfo is valid
6 : OK
7 : OK
8 : OK
9 : OK
10 : OK
11 : confirm()
12 : confirm()
13 : confirm()
14 : tournamentView
15 : tournamentView
16 : tournamentView
17 : sendNewTournamentNotifications()
18 : getNewTournamentInfo()
19 : tournamentInfo
20 : getStudentsList()
21 : getStudentsList()
22 : studentsList
23 : studentsList
loop For all studentID in studentsList
24 : newNotification(studentID, text)
25 : newNotification(studentID, text)
26 : OK
27 : OK
28 : OK

Figure 5: UC1

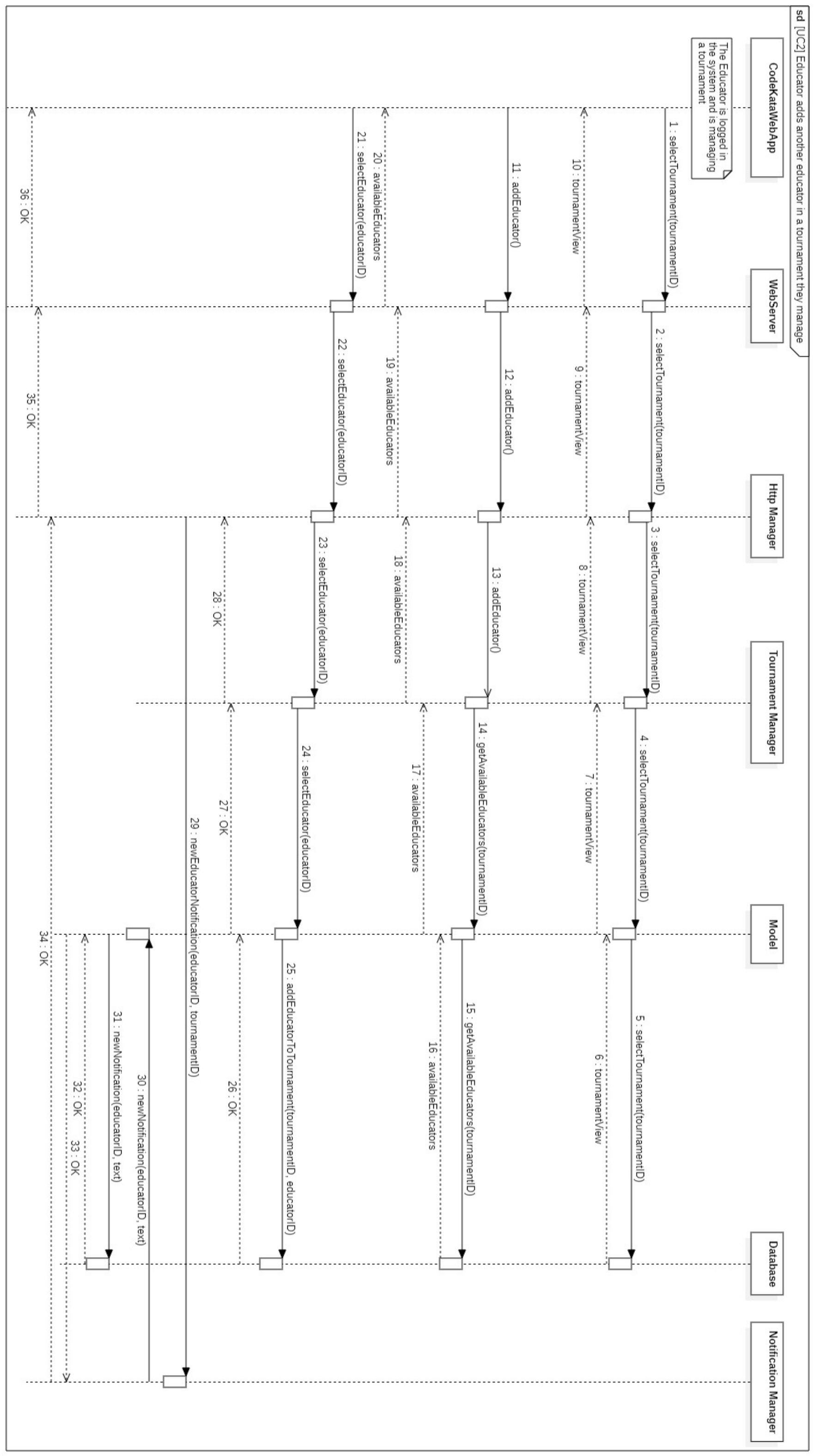

sd [UC2] Educator adds another educator in a tournament they manage
CodeKataWebApp
WebServer
Http Manager
Tournament Manager
Model
Database
Notification Manager
The Educator is logged in the system and is managing a tournament
1 : selectTournament(tournamentID)
2 : selectTournament(tournamentID)
3 : selectTournament(tournamentID)
4 : selectTournament(tournamentID)
5 : selectTournament(tournamentID)
6 : tournamentView
7 : tournamentView
8 : tournamentView
9 : tournamentView
10 : tournamentView
11 : addEducator()
12 : addEducator()
13 : addEducator()
14 : getAvailableEducators(tournamentID)
15 : getAvailableEducators(tournamentID)
16 : availableEducators
17 : availableEducators
18 : availableEducators
19 : availableEducators
20 : availableEducators
21 : selectEducator(educatorID)
22 : selectEducator(educatorID)
23 : selectEducator(educatorID)
24 : selectEducator(educatorID)
25 : addEducatorToTournament(tournamentID, educatorID)
26 : OK
27 : OK
28 : OK
29 : newEducatorNotification(educatorID, tournamentID)
30 : newNotification(educatorID, text)
31 : newNotification(educatorID, text)
32 : OK
33 : OK
34 : OK
35 : OK
36 : OK

Figure 6: UC2

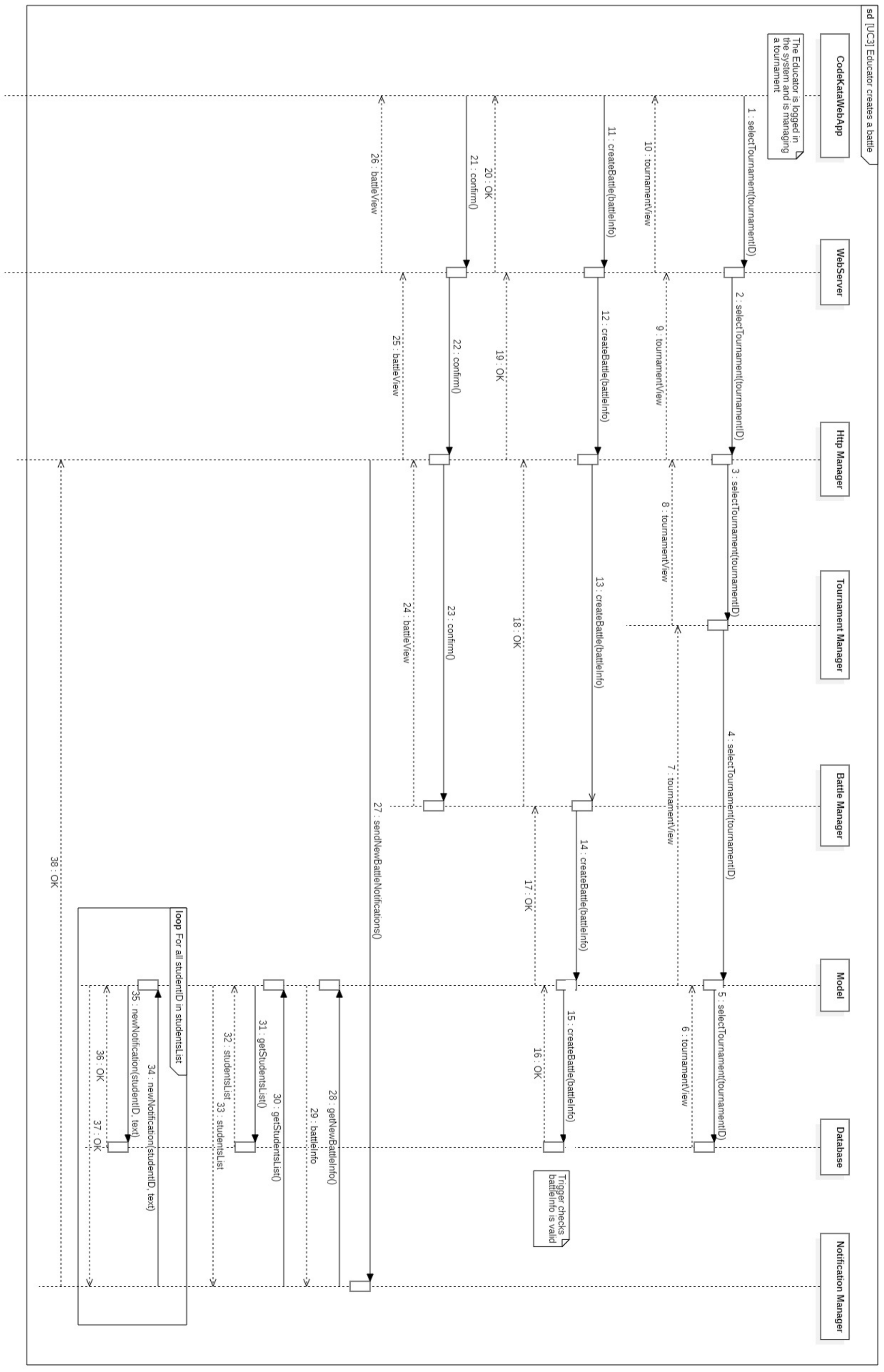

sd [UC3] Educator creates a battle
CodeKataWebApp
WebServer
Http Manager
Tournament Manager
Battle Manager
Model
Database
Notification Manager
The Educator is logged in the system and is managing a tournament
1 : selectTournament(tournamentID)
2 : selectTournament(tournamentID)
3 : selectTournament(tournamentID)
4 : selectTournament(tournamentID)
5 : selectTournament(tournamentID)
6 : tournamentView
7 : tournamentView
8 : tournamentView
9 : tournamentView
10 : tournamentView
11 : createBattle(battleInfo)
12 : createBattle(battleInfo)
13 : createBattle(battleInfo)
14 : createBattle(battleInfo)
15 : createBattle(battleInfo)
16 : OK
17 : OK
18 : OK
19 : OK
20 : OK
Trigger checks battleInfo is valid
21 : confirm()
22 : confirm()
23 : confirm()
24 : battleView
25 : battleView
26 : battleView
27 : sendNewBattleNotifications()
28 : getNewBattleInfo()
29 : battleInfo
30 : getStudentsList()
31 : getStudentsList()
32 : studentsList
33 : studentsList
loop For all studentID in studentsList
34 : newNotification(studentID, text)
35 : newNotification(studentID, text)
36 : OK
37 : OK
38 : OK

Figure 7: UC3

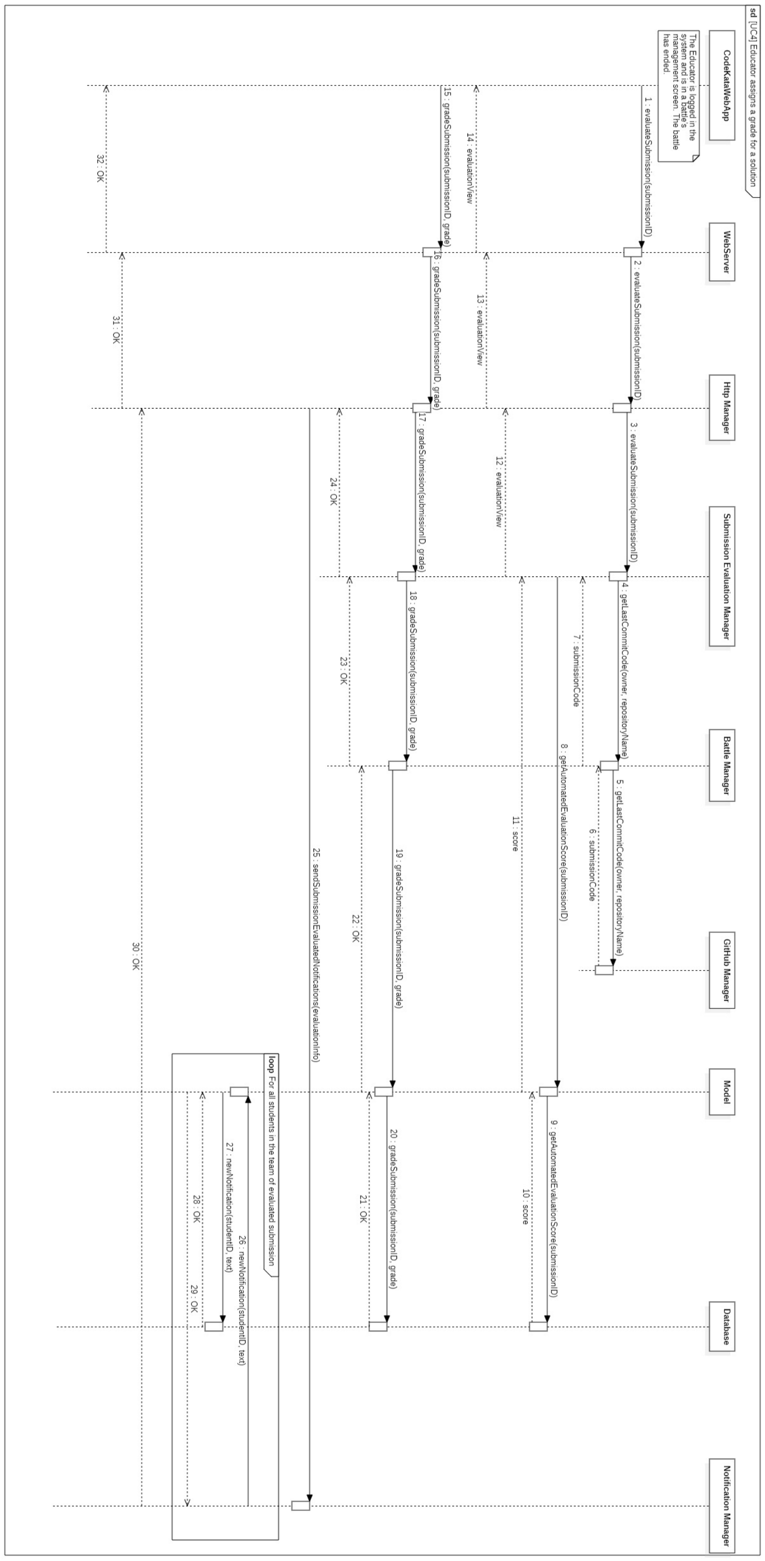

sd [UC4] Educator assigns a grade for a solution
CodeKataWebApp
WebServer
Http Manager
Submission Evaluation Manager
Battle Manager
GitHub Manager
Model
Database
Notification Manager
The Educator is logged in the system and is in a battle's management screen. The battle has ended.
1 : evaluateSubmission(submissionID)
2 : evaluateSubmission(submissionID)
3 : evaluateSubmission(submissionID)
4 : getLastCommitCode(owner, repositoryName)
5 : getLastCommitCode(owner, repositoryName)
6 : submissionCode
7 : submissionCode
8 : getAutomatedEvaluationScore(submissionID)
9 : getAutomatedEvaluationScore(submissionID)
10 : score
11 : score
12 : evaluationView
13 : evaluationView
14 : evaluationView
15 : gradeSubmission(submissionID, grade)
16 : gradeSubmission(submissionID, grade)
17 : gradeSubmission(submissionID, grade)
18 : gradeSubmission(submissionID, grade)
19 : gradeSubmission(submissionID, grade)
20 : gradeSubmission(submissionID, grade)
21 : OK
22 : OK
23 : OK
24 : OK
25 : sendSubmissionEvaluatedNotifications(evaluationInfo)
loop For all students in the team of evaluated submission
26 : newNotification(studentID, text)
27 : newNotification(studentID, text)
28 : OK
29 : OK
30 : OK
31 : OK
32 : OK

Figure 8: UC4

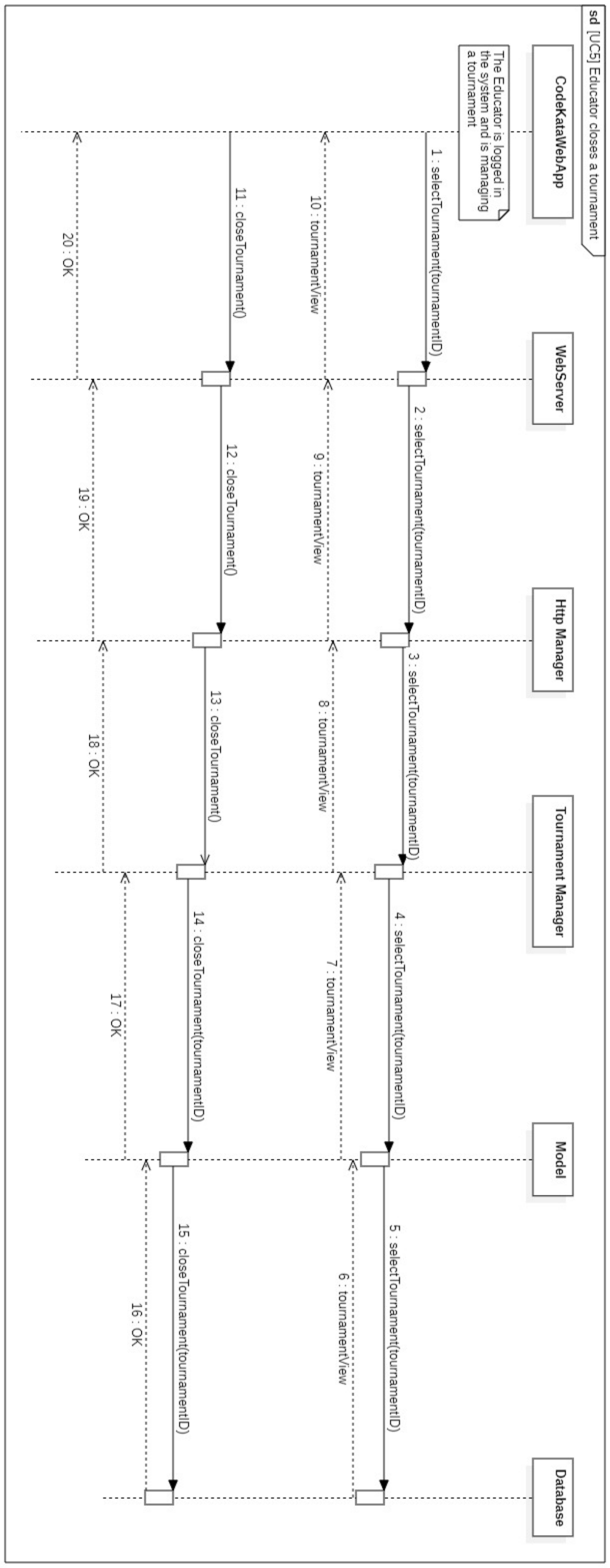

sd [UC5] Educator closes a tournament
CodeKataWebApp
WebServer
Http Manager
Tournament Manager
Model
Database
The Educator is logged in the system and is managing a tournament
1 : selectTournament(tournamentID)
2 : selectTournament(tournamentID)
3 : selectTournament(tournamentID)
4 : selectTournament(tournamentID)
5 : selectTournament(tournamentID)
6 : tournamentView
7 : tournamentView
8 : tournamentView
9 : tournamentView
10 : tournamentView
11 : closeTournament()
12 : closeTournament()
13 : closeTournament()
14 : closeTournament(tournamentID)
15 : closeTournament(tournamentID)
16 : OK
17 : OK
18 : OK
19 : OK
20 : OK

Figure 9: UC5

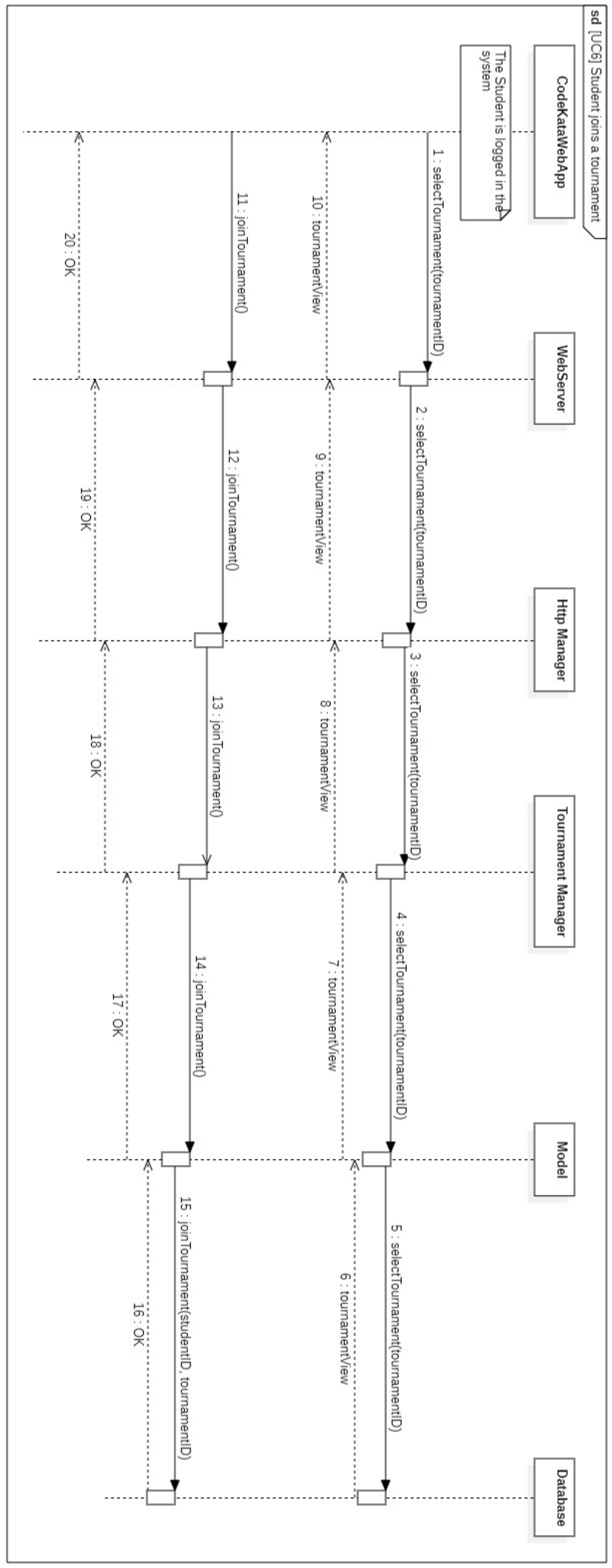

sd [UC6] Student joins a tournament
CodeKataWebApp
WebServer
Http Manager
Tournament Manager
Model
Database
The Student is logged in the system
1 : selectTournament(tournamentID)
2 : selectTournament(tournamentID)
3 : selectTournament(tournamentID)
4 : selectTournament(tournamentID)
5 : selectTournament(tournamentID)
6 : tournamentView
7 : tournamentView
8 : tournamentView
9 : tournamentView
10 : tournamentView
11 : joinTournament()
12 : joinTournament()
13 : joinTournament()
14 : joinTournament()
15 : joinTournament(studentID, tournamentID)
16 : OK
17 : OK
18 : OK
19 : OK
20 : OK

Figure 10: UC6

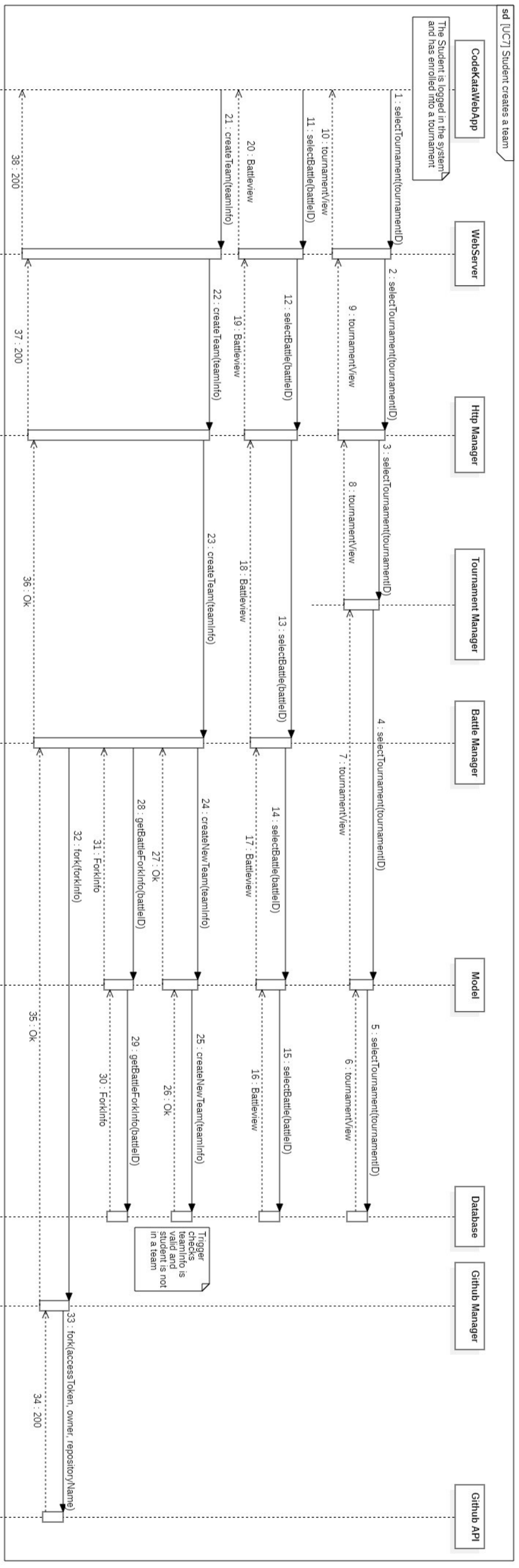

sd [UC7] Student creates a team
CodeKataWebApp
WebServer
Http Manager
Tournament Manager
Battle Manager
Model
Database
Github Manager
Github API
The Student is logged in the system and has enrolled into a tournament
1 : selectTournament(tournamentID)
2 : selectTournament(tournamentID)
3 : selectTournament(tournamentID)
4 : selectTournament(tournamentID)
5 : selectTournament(tournamentID)
6 : tournamentView
7 : tournamentView
8 : tournamentView
9 : tournamentView
10 : tournamentView
11 : selectBattle(battleID)
12 : selectBattle(battleID)
13 : selectBattle(battleID)
14 : selectBattle(battleID)
15 : selectBattle(battleID)
16 : Battleview
17 : Battleview
18 : Battleview
19 : Battleview
20 : Battleview
21 : createTeam(teamInfo)
22 : createTeam(teamInfo)
23 : createTeam(teamInfo)
24 : createNewTeam(teamInfo)
25 : createNewTeam(teamInfo)
26 : Ok
27 : Ok
Trigger checks teamInfo is valid and student is not in a team
28 : getBattleForkInfo(battleID)
29 : getBattleForkInfo(battleID)
30 : ForkInfo
31 : ForkInfo
32 : fork(forkInfo)
33 : fork(accessToken, owner, repositoryName)
34 : 200
35 : Ok
36 : Ok
37 : 200
38 : 200

Figure 11: UC7

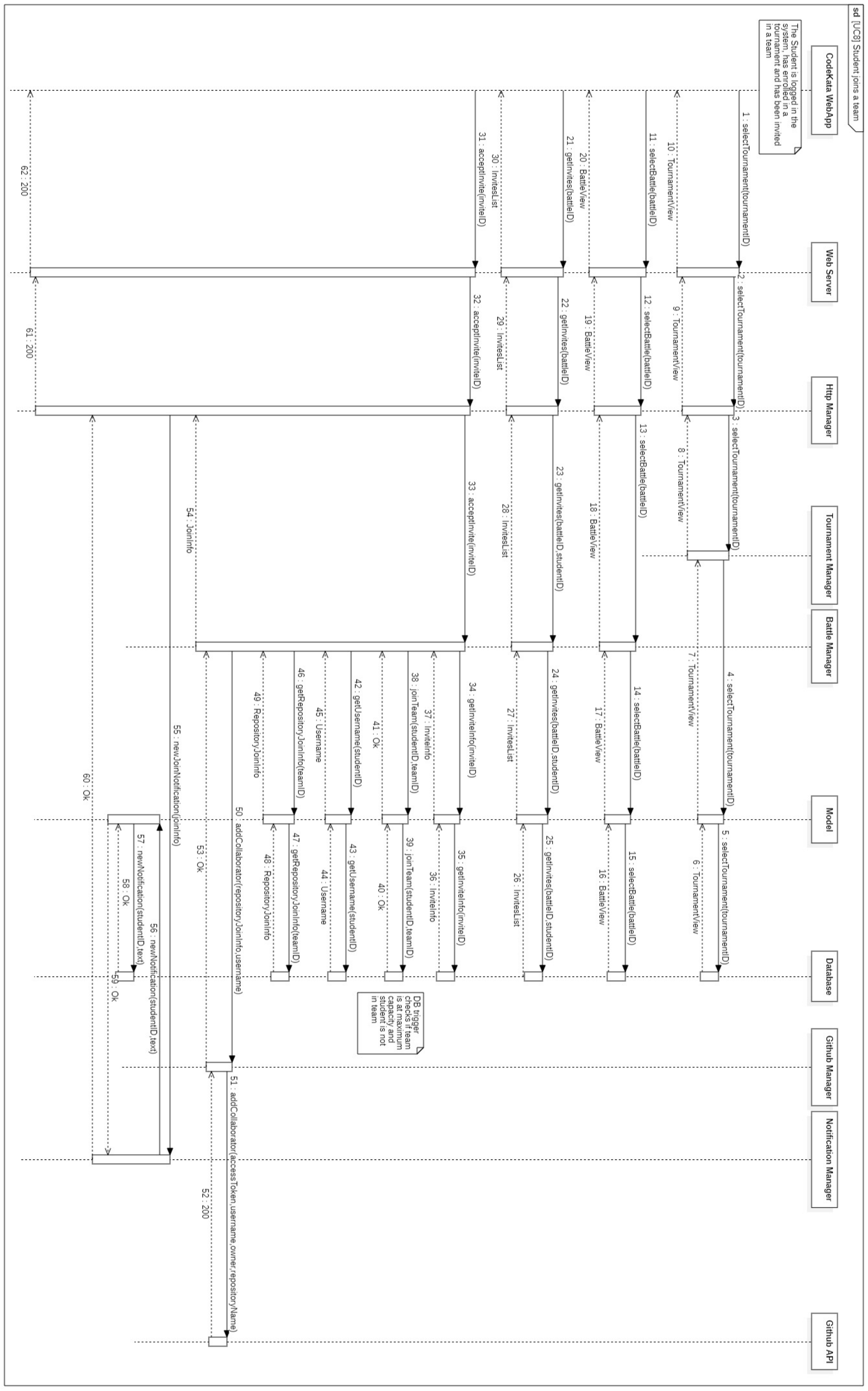

sd [UC8] Student joins a team
CodeKata WebApp
Web Server
Http Manager
Tournament Manager
Battle Manager
Model
Database
Github Manager
Notification Manager
Github API
The Student is logged in the system, has enrolled in a tournament and has been invited in a team
1 : selectTournament(tournamentID)
2 : selectTournament(tournamentID)
3 : selectTournament(tournamentID)
4 : selectTournament(tournamentID)
5 : selectTournament(tournamentID)
6 : TournamentView
7 : TournamentView
8 : TournamentView
9 : TournamentView
10 : TournamentView
11 : selectBattle(battleID)
12 : selectBattle(battleID)
13 : selectBattle(battleID)
14 : selectBattle(battleID)
15 : selectBattle(battleID)
16 : BattleView
17 : BattleView
18 : BattleView
19 : BattleView
20 : BattleView
21 : getInvites(battleID)
22 : getInvites(battleID)
23 : getInvites(battleID,studentID)
24 : getInvites(battleID,studentID)
25 : getInvites(battleID,studentID)
26 : InvitesList
27 : InvitesList
28 : InvitesList
29 : InvitesList
30 : InvitesList
31 : acceptInvite(inviteID)
32 : acceptInvite(inviteID)
33 : acceptInvite(inviteID)
34 : getInviteInfo(inviteID)
35 : getInviteInfo(inviteID)
36 : InviteInfo
37 : InviteInfo
38 : joinTeam(studentID,teamID)
39 : joinTeam(studentID,teamID)
40 : Ok
41 : Ok
DB trigger checks if team is at maximum capacity and student is not in team
42 : getUsername(studentID)
43 : getUsername(studentID)
44 : Username
45 : Username
46 : getRepositoryJoinInfo(teamID)
47 : getRepositoryJoinInfo(teamID)
48 : RepositoryJoinInfo
49 : RepositoryJoinInfo
50 : addCollaborator(repositoryJoinInfo,username)
51 : addCollaborator(accessToken,username,owner,repositoryName)
52 : 200
53 : Ok
54 : JoinInfo
55 : newJoinNotification(joinInfo)
56 : newNotification(studentID,text)
57 : newNotification(studentID,text)
58 : Ok
59 : Ok
60 : Ok
61 : 200
62 : 200

Figure 12: UC8

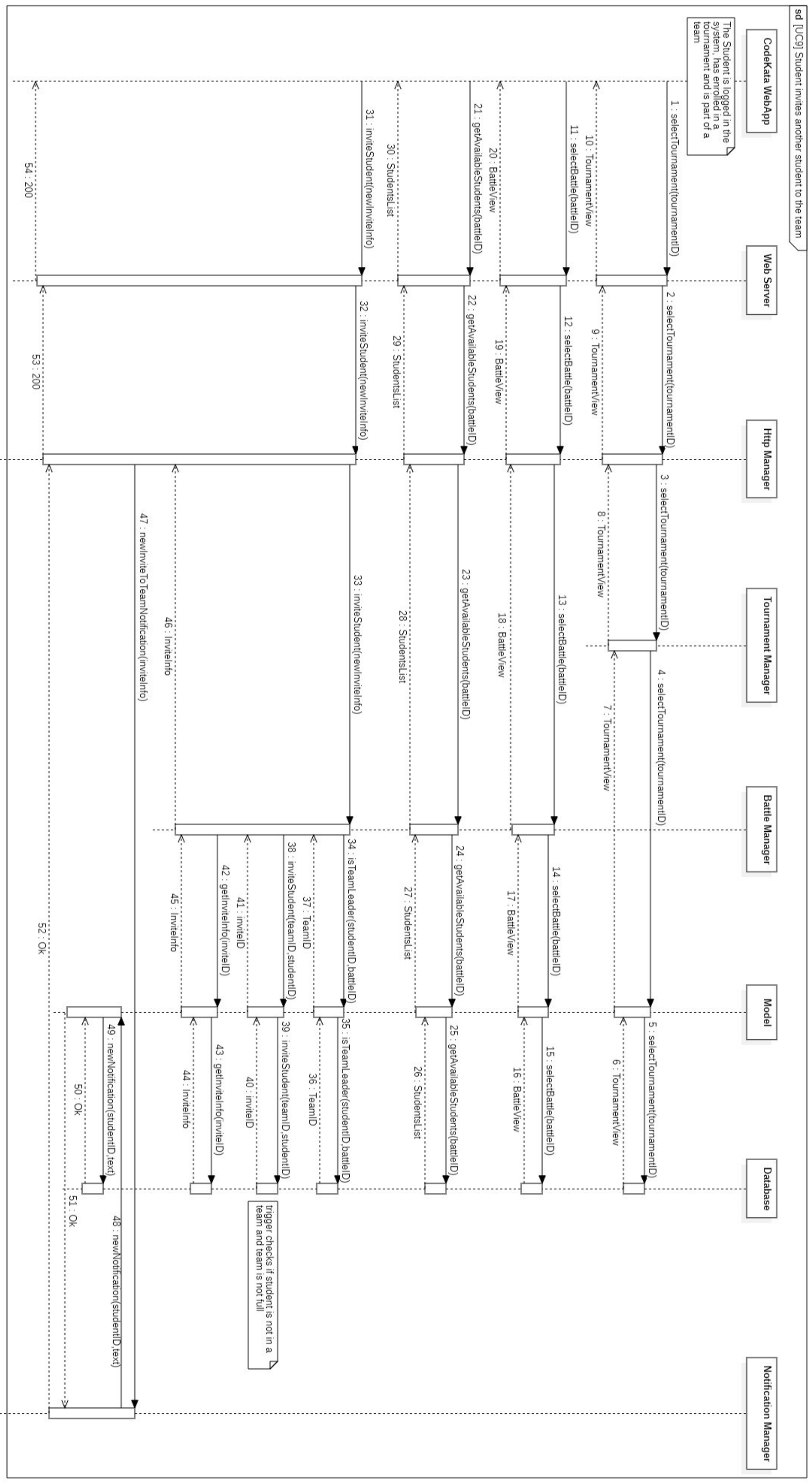

sd [UC9] Student invites another student to the team
CodeKata WebApp
Web Server
Http Manager
Tournament Manager
Battle Manager
Model
Database
Notification Manager
The Student is logged in the system, has enrolled in a tournament and is part of a team
1 : selectTournament(tournamentID)
2 : selectTournament(tournamentID)
3 : selectTournament(tournamentID)
4 : selectTournament(tournamentID)
5 : selectTournament(tournamentID)
6 : TournamentView
7 : TournamentView
8 : TournamentView
9 : TournamentView
10 : TournamentView
11 : selectBattle(battleID)
12 : selectBattle(battleID)
13 : selectBattle(battleID)
14 : selectBattle(battleID)
15 : selectBattle(battleID)
16 : BattleView
17 : BattleView
18 : BattleView
19 : BattleView
20 : BattleView
21 : getAvailableStudents(battleID)
22 : getAvailableStudents(battleID)
23 : getAvailableStudents(battleID)
24 : getAvailableStudents(battleID)
25 : getAvailableStudents(battleID)
26 : StudentsList
27 : StudentsList
28 : StudentsList
29 : StudentsList
30 : StudentsList
31 : inviteStudent(newInviteInfo)
32 : inviteStudent(newInviteInfo)
33 : inviteStudent(newInviteInfo)
34 : isTeamLeader(studentID,battleID)
35 : isTeamLeader(studentID,battleID)
36 : TeamID
37 : TeamID
38 : inviteStudent(teamID,studentID)
39 : inviteStudent(teamID,studentID)
40 : inviteID
41 : inviteID
42 : getInviteInfo(inviteID)
43 : getInviteInfo(inviteID)
44 : InviteInfo
45 : InviteInfo
46 : InviteInfo
47 : newInviteToTeamNotification(inviteInfo)
48 : newNotification(studentID,text)
49 : newNotification(studentID,text)
50 : Ok
51 : Ok
52 : Ok
53 : 200
54 : 200
trigger checks if student is not in a team and team is not full

Figure 13: UC9

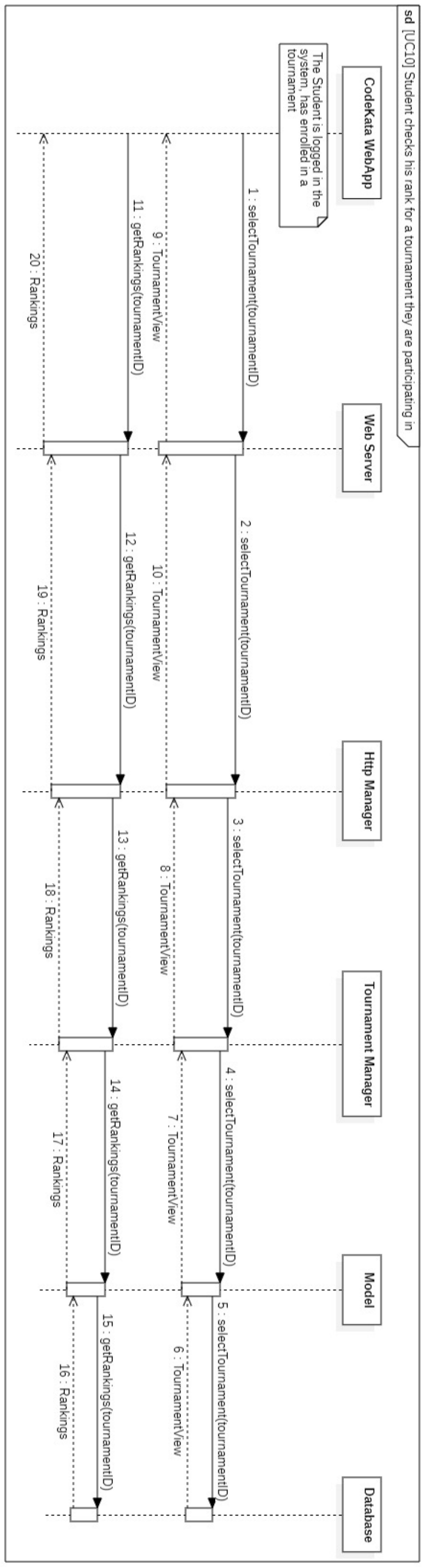

sd [UC10] Student checks his rank for a tournament they are participating in
CodeKata WebApp
Web Server
Http Manager
Tournament Manager
Model
Database
The Student is logged in the system, has enrolled in a tournament
1 : selectTournament(tournamentID)
2 : selectTournament(tournamentID)
3 : selectTournament(tournamentID)
4 : selectTournament(tournamentID)
5 : selectTournament(tournamentID)
6 : TournamentView
7 : TournamentView
8 : TournamentView
9 : TournamentView
10 : TournamentView
11 : getRankings(tournamentID)
12 : getRankings(tournamentID)
13 : getRankings(tournamentID)
14 : getRankings(tournamentID)
15 : getRankings(tournamentID)
16 : Rankings
17 : Rankings
18 : Rankings
19 : Rankings
20 : Rankings

Figure 14: UC10

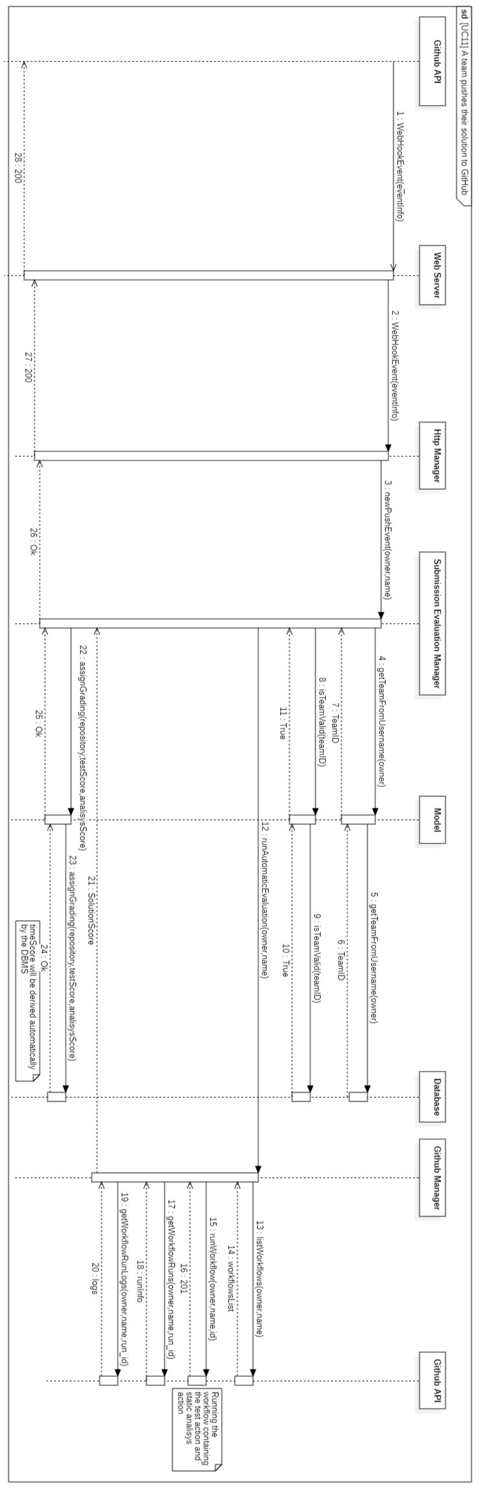

sd [UC11] A team pushes their solution to GitHub
Github API
Web Server
Http Manager
Submission Evaluation Manager
Model
Database
Github Manager
Github API
1 : WebHookEvent(eventInfo)
2 : WebHookEvent(eventInfo)
3 : newPushEvent(owner,name)
4 : getTeamFromUsername(owner)
5 : getTeamFromUsername(owner)
6 : TeamID
7 : TeamID
8 : isTeamValid(teamID)
9 : isTeamValid(teamID)
10 : True
11 : True
12 : runAutomaticEvaluation(owner,name)
13 : listWorkflows(owner,name)
14 : workflowsList
15 : runWorkflow(owner,name,id)
16 : 201
17 : getWorkflowRuns(owner,name,run_id)
18 : runInfo
19 : getWorkflowRunLogs(owner,name,run_id)
20 : logs
21 : SolutionScore
22 : assignGrading(repository,testScore,analisysScore)
23 : assignGrading(repository,testScore,analisysScore)
24 : Ok
25 : Ok
26 : Ok
27 : 200
28 : 200
timeScore will be derived automatically by the DBMS
Running the workflow containing the test action and static analisys action

Figure 15: UC11

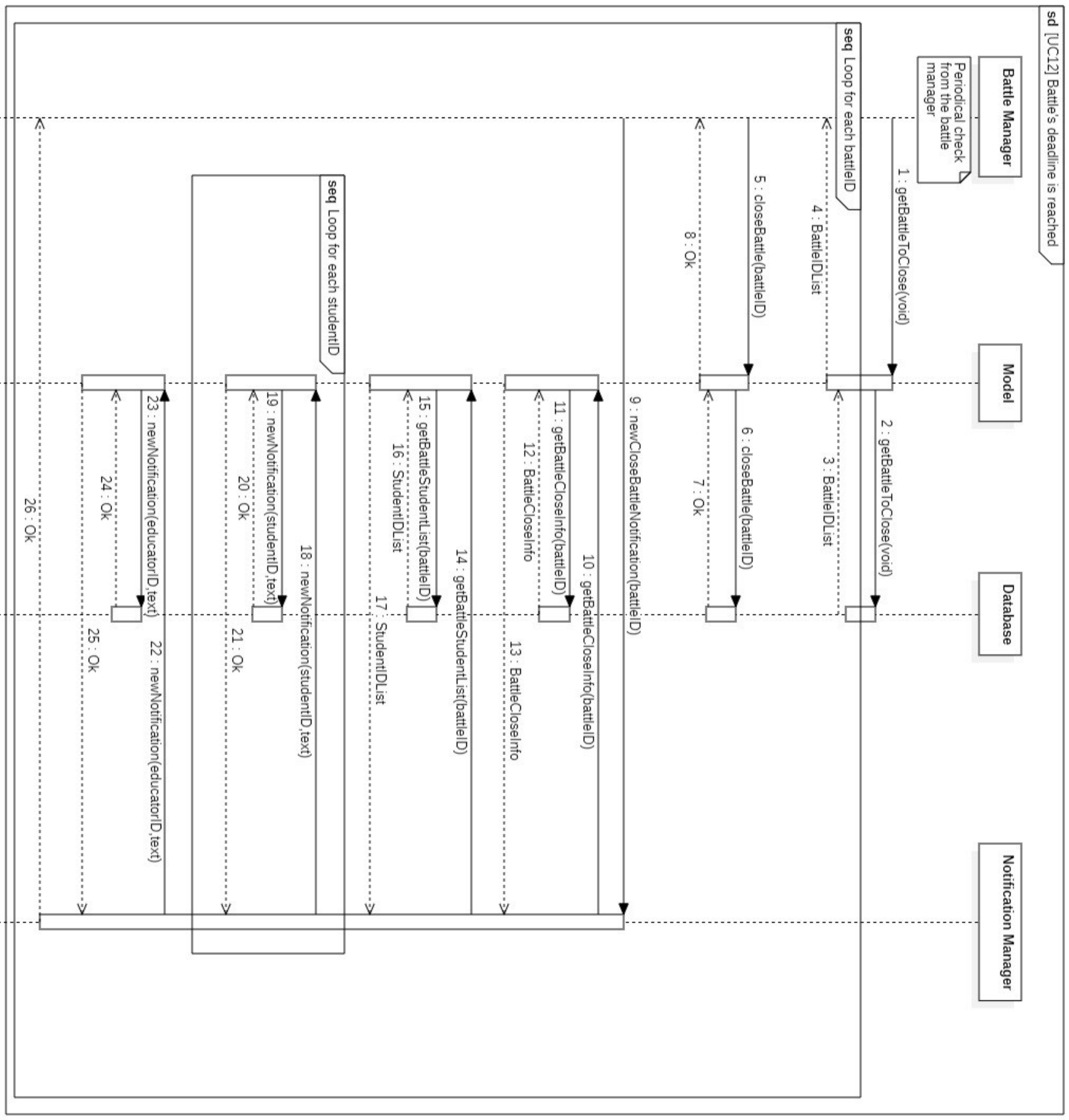

sd [UC12] Battle's deadline is reached
Battle Manager
Model
Database
Notification Manager
Periodical check from the battle manager
1 : getBattleToClose(void)
2 : getBattleToClose(void)
3 : BattleIDList
4 : BattleIDList
seq Loop for each battleID
5 : closeBattle(battleID)
6 : closeBattle(battleID)
7 : Ok
8 : Ok
9 : newCloseBattleNotification(battleID)
10 : getBattleCloseInfo(battleID)
11 : getBattleCloseInfo(battleID)
12 : BattleCloseInfo
13 : BattleCloseInfo
14 : getBattleStudentList(battleID)
15 : getBattleStudentList(battleID)
16 : StudentIDList
17 : StudentIDList
seq Loop for each studentID
18 : newNotification(studentID,text)
19 : newNotification(studentID,text)
20 : Ok
21 : Ok
22 : newNotification(educatorID,text)
23 : newNotification(educatorID,text)
24 : Ok
25 : Ok
26 : Ok

Figure 16: UC12

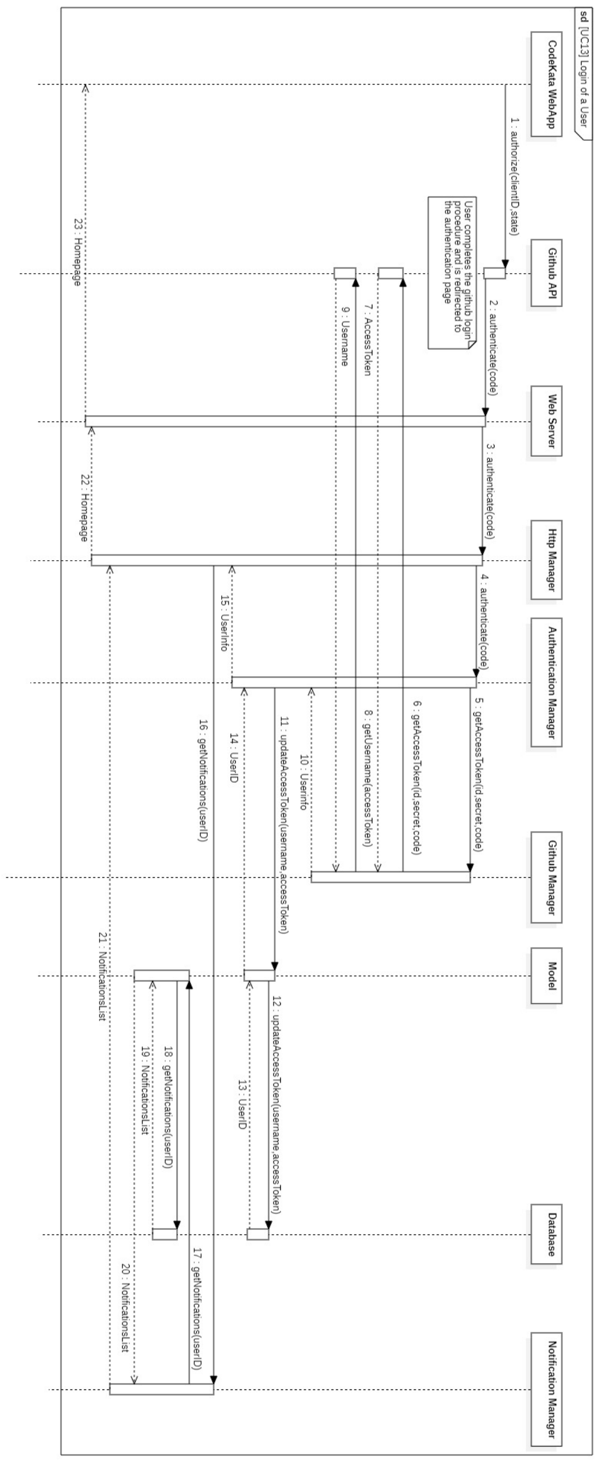

sd [UC13] Login of a User
CodeKata WebApp
Github API
Web Server
Http Manager
Authentication Manager
Github Manager
Model
Database
Notification Manager
1 : authorize(clientID,state)
User completes the github login procedure and is redirected to the authentication page
2 : authenticate(code)
3 : authenticate(code)
4 : authenticate(code)
5 : getAccessToken(id,secret,code)
6 : getAccessToken(id,secret,code)
7 : AccessToken
8 : getUsername(accessToken)
9 : Username
10 : UserInfo
11 : updateAccessToken(username,accessToken)
12 : updateAccessToken(username,accessToken)
13 : UserID
14 : UserID
15 : UserInfo
16 : getNotifications(userID)
17 : getNotifications(userID)
18 : getNotifications(userID)
19 : NotificationsList
20 : NotificationsList
21 : NotificationsList
22 : Homepage
23 : Homepage

Figure 17: UC13

2.5 Component interfaces

The Github API interfaces used in the diagrams are not listed as they are just a simplification over the real Github API endpoint

2.5.1 Web Server

createTournament(tournamentInfo)
Rest API call used to create a new tournament
Param: info about the new tournament
Returns: 200 if successful

selectTournament(tournamentID)
Rest API call used to get the tournament view
Param: tournamentID the ID of the tournament
Returns: all data needed to build the tournament view if successful

addEducator()
Rest API call used to get the list of all the educators that can be added to the current tournament
Returns: List of available educators

selectEducator(educatorID)
Rest API call to add an educator to a tournament
Param: educatorID
Returns: Ok value if successful

createBattle(battleInfo)
Rest API call used to create a battle
Param: info about the battle to be created
Returns: Ok value if successful

gradeSubmission(submissionID, grade)
Rest API call used to assign a grade to a specified submission
Param: submissionID ID of the submission to grade
Grade: grade to be assigned to the submission
Returns: Ok value if successful

closeTournament()
Rest API call used to close the currently selected tournament
Returns: Ok value if successful

selectBattle(battleID)
Rest API call used to get the battle
Param: battleID the ID of the battle
Returns: all data needed to build the battle view if successful

createTeam(teamInfo)
Rest API call used create a new team for a battle
Param: teamInfo the info about the new team
Returns: 200 if successful

getInvites(battleID)
Rest API call used to get the invites for the specified battle
Param: battleID the ID of the battle
Returns: list of invites and their id if successful

acceptInvite(inviteID)
Rest API call used to accept an invite to a team
Param: inviteID the id of the invite
Returns: 200 if successful

getAvailableStudents(battleID)
Rest API call used to get the available students for the specified battle
Param: battleID the ID of the battle
Returns: list of students usernames and their id if successful

inviteStudent(newInviteInfo)
Rest API call used to invite a new student
Param: newInviteInfo info regarding the student being invited and the battle
Returns: 200 if successful

getRankings(tournamentID)
Rest API call used to get the rankings of the tournament
Param: tournamentID the ID of the tournament
Returns: a list of username and their score if successful

WebHookEvent(eventInfo)
Rest API call used by github to notify a new version of a repository of a user has been pushed
Param: eventInfo Info regarding the webhook event as stated in the github api documentation
Returns: 200 if successful

authenticate(code)
Rest API callback used by github to redirect a user after they completed authentication
Param: code the code needed to create a new access code for the user
Returns: The Homepage if successful

evaluateSubmission(submissionID)
Rest API called to show the evaluation view for the selected submission
Param: submissionID of the submission to be evaluated
Returns: evaluation view for the selected submission

joinTournament()
Rest API call used to make the current student join the selected tournament
Returns: Ok value if successful

2.5.2 HTTP Manager

createTournament(tournamentInfo)
Rest API call used to create a new tournament
Param: info about the new tournament
Returns: 200 if successful

selectTournament(tournamentID)
Rest API call used to get the tournament view
Param: tournamentID the ID of the tournament
Returns: all data needed to build the tournament view if successful

addEducator()

Rest API call used to get the list of all the educators that can be added to the current tournament
Returns: List of available educators

selectEducator(educatorID)
Rest API call to add an educator to a tournament
Param: educatorID
Returns: Ok value if successful

createBattle(battleInfo)
Rest API call used to create a battle
Param: info about the battle to be created
Returns: Ok value if successful

selectBattle(battleID)
Rest API call used to get the battle
Param: battleID the ID of the battle
Returns: all data needed to build the battle view if successful

createTeam(teamInfo)
Rest API call used create a new team for a battle
Param: teamInfo the info about the new team
Returns: 200 if successful

getInvites(battleID)
Rest API call used to get the invites for the specified battle
Param: battleID the ID of the battle
Returns: list of invites and their id if successful

acceptInvite(inviteID)
Rest API call used to accept an invite to a team
Param: inviteID the id of the invite
Returns: 200 if successful

getAvailableStudents(battleID)
Rest API call used to get the available students for the specified battle
Param: battleID the ID of the battle
Returns: list of students usernames and their id if successful

inviteStudent(newInviteInfo)
Rest API call used to invite a new student
Param: newInviteInfo info regarding the student being invited and the battle
Returns: 200 if successful

getRankings(tournamentID)
Rest API call used to get the rankings of the tournament
Param: tournamentID the ID of the tournament
Returns: a list of username and their score if successful

WebHookEvent(eventInfo)
Rest API call used by github to notify a new version of a repository of a user has been pushed
Param: eventInfo Info regarding the webhook event as stated in the github api documentation
Returns: 200 if successful

authenticate(code)

Rest API callback used by github to redirect a user after they completed authentication
Param: code the code needed to create a new access code for the user
Returns: The Homepage if successful

evaluateSubmission(submissionID)
Rest API called to show the evaluation view for the selected submission
Param: submissionID of the submission to be evaluated
Returns: evaluation view for the selected submission

gradeSubmission(submissionID, grade)
Rest API call used to assign a grade to a specified submission
Param: submissionID ID of the submission to grade
Grade: grade to be assigned to the submission
Returns: Ok value if successful

closeTournament()
Rest API call used to close the currently selected tournament
Returns: Ok value if successful

joinTournament()
Rest API call used to make the current student join the selected tournament
Returns: Ok value if successful

2.5.3 Tournament Manager

createTournament(tournamentInfo)
Function call used to create a new tournament
Param: info about the new tournament
Returns: 200 if successful

selectTournament(tournamentID)
Function call used to get the tournament view
Param: tournamentID the ID of the tournament
Returns: all data needed to build the tournament view if successful

addEducator()
Rest API call used to get the list of all the educators that can be added to the current tournament
Returns: List of available educators

selectEducator(educatorID)
Function call to add an educator to a tournament
Param: educatorID
Returns: Ok value if successful

closeTournament()
Function call used to close the currently selected tournament
Returns: Ok value if successful

joinTournament()
Function call used to make the current student join the selected tournament
Returns: Ok value if successful

getRankings(tournamentID)

Function call used to get the rankings of the tournament
Param: tournamentID the ID of the tournament
Returns: a list of username and their score if successful

2.5.4 Battle Manager

createBattle(battleInfo)
Function call used to create a battle
Param: info about the battle to be created
Returns: Ok value if successful

selectBattle(battleID)
Function call used to get the battle
Param: battleID the ID of the battle
Returns: all data needed to build the battle view if successful

createTeam(teamInfo)
Function call used create a new team for a battle
Param: teamInfo the info about the new team
Returns: Ok value if successful

getBattleForkInfo(forkInfo)
Function call used get all info needed to fork the battle repository
Param: battleID the id of the battle related to the repository
Returns: the fork informations if successful

getInvites(battleID,studentID)
Function call used to get the invites for the specified battle
Param: battleID the ID of the battle
Param: studentID the ID of the student invited
Returns: list of invites and their id if successful

acceptInvite(inviteID)
Function call used to accept an invite to a team
Param: inviteID the id of the invite
Returns: Info regarding the student that has joined and the team that has been joined if successful

getAvailableStudents(battleID)
Function call used to get the available students for the specified battle
Param: battleID the ID of the battle
Returns: list of students usernames and their id if successful

inviteStudent(newInviteInfo)
Function call used to invite a new student
Param: newInviteInfo info regarding the student being invited, the battle and the student that has invited
Returns: Info regarding the student that should join and the team that should be joined if successful

getLastCommitCode(owner, repositoryName)
Function call used to get the code of the last submission of the specified repository
Param: owner
repositoryName: name of the repository
Returns: code of the last submission of the repository

gradeSubmission(submissionID, grade)
Function call used to assign a grade to a specified submission
Param: submissionID ID of the submission to grade
Grade: grade to be assigned to the submission
Returns: Ok value if successful

2.5.5 Model

createTournament(tournamentInfo)
Function call used to create a new tournament
Param: info about the new tournament
Returns: 200 if successful

selectTournament(tournamentID)
Function call used to get the tournament view
Param: tournamentID the ID of the tournament
Returns: all data needed to build the tournament view if successful

getNewTournamentInfo()
Function call used to get the info about the newly created tournament
Returns: tournamentInfo

getNewBattleInfo()
Function call used to get the info about the newly created battle
Returns: battleInfo

getStudentsList()
Function call used to get the list of all the students subscribed to the CKB platform
Returns: Students list

getAvailableEducators(tournamentID)
Function call used to get the list of all the educators that can be added to the specified tournament
Param: tournamentID the ID of the tournament
Returns: List of available educators

selectEducator(educatorID)
Function call to add an educator to a tournament
Param: educatorID
Returns: Ok value if successful

createBattle(battleInfo)
Function call used to create a battle
Param: info about the battle to be created
Returns: Ok value if successful

getAutomatedEvaluationScore(submissionID)
Function call used to get the score of the automatic evaluation of the specified submission
Param: submissionID
Returns: score of the automatic evaluation of the submission

closeTournament(tournamentID)
Function call used to close the currently selected tournament
Param: ID of the tournament to be closed

Returns: Ok value if successful

joinTournament()
Function call used to make the current student join the selected tournament
Returns: Ok value if successful

selectBattle(battleID)
Function call used to get the battle
Param: battleID the ID of the battle
Returns: all data needed to build the battle view if successful

createTeam(teamInfo)
Function call used create a new team for a battle
Param: teamInfo the info about the new team
Returns: Ok value if successful

getBattleForkInfo(forkInfo)
Function call used get all info needed to fork the battle repository
Param: battleID the id of the battle related to the repository
Returns: the fork informations if successful

getInvites(battleID,studentID)
Function call used to get the invites for the specified battle
Param: battleID the ID of the battle
Param: studentID the ID of the student invited
Returns: list of invites and their id if successful

getInviteInfo(inviteID)
Function call used to get info about an invite
Param: inviteID the id of the invite
Returns: Info regarding the student that should join and the team that should be joined if successful

joinTeam(studentID,teamID)
Function call used to make a student join a team
Param: studentID the id of the student that is joining
Param: teamID the id of the team
Returns: Ok value if successful

getUsername(studentID)
Function call used to get the github username of a student
Param: studentID the id of the student
Returns: a username if successful

getRepositoryJoinInfo(teamID)
Function call used to get the info needed by the github api to add a collaborator
Param: teamID the id of the team
Returns: the info needed if successful

newNotification(studentID,text)
Function call used to create a new notification
Param: studentID the student to which the notification is directed
Param: text the text of the notification
Returns: Ok value if successful

newNotification(educatorID,text)
Function call used to create a new notification
Param: educatorID the educator to which the notification is directed
Param: text the text of the notification
Returns: Ok value if successful

getAvailableStudents(battleID)
Function call used to get the available students for the specified battle
Param: battleID the ID of the battle
Returns: list of students usernames and their id if successful

isTeamLeader(studentID,battleID)
Function call used to check if a student is team leader
Param: studentID the id of the student
Param: battleID the id of the battle
Returns: the id of the team if the student is team leader if successful

inviteStudent(teamID,studentID)
Function call used to invite a student to a team
Param: teamID the id of the team
Param: studentID the id of the student to invite
Returns: the id of the invite generated if successful

getRankings(tournamentID)
Function call used to get the rankings of the tournament
Param: tournamentID the ID of the tournament
Returns: a list of username and their score if successful

getTeamFromUsername(owner)
Function call used to get the teamID of the owner of a repository if the repository is linked to a battle
Param: owner the username of the repository
Returns: the teamID of the team if successful

isTeamValid(teamID)
Function call used to check if the team meets the requirements of the battle to compete and if the battle is open
Param: teamID the ID of the team to check
Returns: true if the conditions are met

assignGrading(repository,testScore,analysisScore)
Function call used to set the score of a team for the automatic grading
Param: repository the name of the repository
Param: testScore the score regarding the testing
Param: analysisScore the score regarding the static analysis
Returns: Ok value if successful

getBattleToClose()
Function call used to get a list of all the battle that need to be closed
Returns: a list of battles if successful

closeBattle(battleID)
Function call used to set a battle to closed status
Param: battleID the ID of the battle
Returns: Ok value if successful

newNotification(educatorID,text)
Function call used to create a new notification
Param: educatorID the educator to which the notification is directed
Param: text the text of the notification
Returns: Ok value if successful

getBattleCloseInfo(battleID)
Function call used to get information about the name of a battle and its creator
Param: battleID the ID of the battle
Returns: information about the name of a battle and its creator if successful

getBattleStudentList(battleID)
Function call used to get the list of students participating in a battle
Param: battleID the ID of the battle
Returns: a list of studentID if successful

getNotifications(userID)
Function call used to get all the new notification of a user
Param: userID the ID of the user
Returns: list of notifications if successful

updateAccessToken(username)
Function call used to update the user access token and get the userID
Param: username the github username of the user
Returns: the userID if successful

gradeSubmission(submissionID, grade)
Function call used to assign a grade to a specified submission
Param: submissionID ID of the submission to grade
Param: grade to be assigned to the submission
Returns: Ok value if successful

2.5.6 Database

All database api call refer to prepared statements in the SQL database **createTournament(tournamentInfo)**
Database API call used to create a new tournament
Param: info about the new tournament
Returns: 200 if successful

selectTournament(tournamentID)
Database API call used to get the tournament view
Param: tournamentID the ID of the tournament
Returns: all data needed to build the tournament view if successful

getStudentsList()
Database API call used to get the list of all the students subscribed to the CKB platform
Returns: Students list

getAvailableEducators(tournamentID)
Database API call used to get the list of all the educators that can be added to the specified tournament
Param: tournamentID the ID of the tournament
Returns: List of available educators

addEducatorToTournament(tournamentID, educatorID)
Database API call used to add an Educator to a tournament
Param: tournamentID
educatorID: id of the educator to be added to the tournament
Returns: Ok value if successful

createBattle(battleInfo)
Database API call used to create a battle
Param: info about the battle to be created
Returns: Ok value if successful

getAutomatedEvaluationScore(submissionID)
Database API call used to get the score of the automatic evaluation of the specified submission
Param: submissionID
Returns: score of the automatic evaluation of the submission

gradeSubmission(submissionID, grade)
Database API call used to assign a grade to a specified submission
Param: submissionID ID of the submission to grade
Grade: grade to be assigned to the submission
Returns: Ok value if successful

closeTournament(tournamentID)
Database API call used to close the currently selected tournament
Param: ID of the tournament to be closed
Returns: Ok value if successful

joinTournament(studentID, tournamentID)
Database API call used to make a student join the specified tournament
Param: studentID
tournamentID: ID of the tournament
Returns: Ok value if successful

selectBattle(battleID)
Database API call used to get the battle
Param: battleID the ID of the battle
Returns: all data needed to build the battle view if successful

createTeam(teamInfo)
Database API call used create a new team for a battle
Param: teamInfo the info about the new team
Returns: Ok value if successful

getBattleForkInfo(forkInfo)
Database API call used get all info needed to fork the battle repository
Param: battleID the id of the battle related to the repository
Returns: the fork informations if successful

getInvites(battleID,studentID)
Database API call used to get the invites for the specified battle
Param: battleID the ID of the battle
Param: studentID the ID of the student invited
Returns: list of invites and their id if successful

getInviteInfo(inviteID)
Database API call used to get info about an invite
Param: inviteID the id of the invite
Returns: Info regarding the student that should join and the team that should be joined if successful

joinTeam(studentID,teamID)
Database API call used to make a student join a team
Param: studentID the id of the student that is joining
Param: teamID the id of the team
Returns: Ok value if successful

getUsername(studentID)
Database API call used to get the github username of a student
Param: studentID the id of the student
Returns: a username if successful

getRepositoryJoinInfo(teamID)
Database API call used to get the info needed by the github api to add a collaborator
Param: teamID the id of the team
Returns: the info needed if successful

newNotification(studentID,text)
Database API call used to create a new notification
Param: studentID the student to which the notification is directed
Param: text the text of the notification
Returns: Ok value if successful

getAvailableStudents(battleID)
Database API call used to get the available students for the specified battle
Param: battleID the ID of the battle
Returns: list of students usernames and their id if successful

isTeamLeader(studentID,battleID)
Database API call used to check if a student is team leader
Param: studentID the id of the student
Param: battleID the id of the battle
Returns: the id of the team if the student is team leader if successful

inviteStudent(teamID,studentID)
Database API call used to invite a student to a team
Param: teamID the id of the team
Param: studentID the id of the student to invite
Returns: the id of the invite generated if successful

getRankings(tournamentID)
Database API call used to get the rankings of the tournament
Param: tournamentID the ID of the tournament
Returns: a list of username and their score if successful

getTeamFromUsername(owner)
Database API call used to get the teamID of the owner of a repository if the repository is linked to a battle
Param: owner the username of the repository
Returns: the teamID of the team if successful

isTeamValid(teamID)
Database API call used to check if the team meets the requirements of the battle to compete and if the battle is open
Param: teamID the ID of the team to check
Returns: true if the conditions are met

assignGrading(repository,testScore,analysisScore)
Database API used to set the score of a team for the automatic grading
Param: repository the name of the repository
Param: testScore the score regarding the testing
Param: analysisScore the score regarding the static analysis
Returns: Ok value if successful

getBattleToClose()
Database API call used to get a list of all the battle that need to be closed
Returns: a list of battles if successful

closeBattle(battleID)
Database API call used to set a battle to closed status
Param: battleID the ID of the battle
Returns: Ok value if successful

newNotification(educatorID,text)
Database API call used to create a new notification
Param: educatorID the educator to which the notification is directed
Param: text the text of the notification
Returns: Ok value if successful

getBattleCloseInfo(battleID)
Database API call used to get information about the name of a battle and its creator
Param: battleID the ID of the battle
Returns: information about the name of a battle and its creator if successful

getBattleStudentList(battleID)
Database API call used to get the list of students participating in a battle
Param: battleID the ID of the battle
Returns: a list of studentID if successful

getNotifications(userID)
Database API call used to get all the new notification of a user
Param: userID the ID of the user
Returns: list of notifications if successful

updateAccessToken(username)
Database API call used to update the user access token and get the userID
Param: username the github username of the user
Returns: the userID if successful

2.5.7 Github Manager

fork(forkInfo)
Function call used to fork a repository

Param: forkInfo all info needed by the github api to fork a repository
Returns: Ok value if successful

addCollaborator(repositoryJoinInfo,username)
Function call used to add a collaborator to a github repository
Param: repositoryJoinInfo the info needed by the github api to add a collaborator
Param: username the username of the collaborator to add
Returns: Ok value if successful

runAutomaticEvaluation(owner,name)
Function call used to set the score of a team for the automatic grading
Param: owner the username of the owner of the repository
Param: name the name of the repository
Returns: The result of the evaluation if successful

getAccessToken(id,secret,code)
Function call used to get a new access code for a user
Param: id the id of the github app used by the system
Param: secret the secret key of the system give by github
Param: code the code given by the authorization request
Returns: The access token and the user username if successful

getLastCommitCode(owner, repositoryName)
Function call used to get the code of the last submission of the specified repository
Param: owner
repositoryName: name of the repository
Returns: code of the last submission of the repository

2.5.8 Notification Manager

sendNewTournamentNotifications()
Function call used to send a new tournament notification to all Students
Returns: Ok value if successful

newEducatorNotification(educatorID, tournamentID)
Function call used to notify an educator that they have been added to a tournament
Param: educatorID
tournamentID: ID of the tournament the educator has been added to
Returns: Ok value if successful

sendNewBattleNotifications()
Function call used to send a new battle notification to all Students
Returns: Ok value if successful

sendSubmissionEvaluatedNotifications(evaluationInfo)
Function call used to send a notification to the students of a team about the evaluation of their submission
Param: info regarding the submission and its grade
Returns: Ok value if successful

newJoinNotification(joinInfo)
Function call used to create a new join notification
Param: joinInfo Info regarding the student that has joined and the team that has been joined
Returns: Ok value if successful

newInviteToTeamNotification(inviteInfo)
Function call used to create a new join notification
Param: inviteInfo Info regarding the student that should join and the team that should be joined
Returns: Ok value if successful

newCloseBattleNotification(battleID)
Function call used to create notifications for all the student of a battle and its creator when the battle is closed
Param: battleID the ID of the battle
Returns: Ok value if successful

getNotifications(userID)
Function call used to get all the new notification of a user
Param: userID the ID of the user
Returns: list of notifications if successful

2.5.9 Submission Evaluation Manager

evaluateSubmission(submissionID)
Function called to show the evaluation view for the selected submission
Param: submissionID of the submission to be evaluated
Returns: evaluation view for the selected submission

gradeSubmission(submissionID, grade)
Function call used to assign a grade to a specified submission
Param: submissionID ID of the submission to grade
Grade: grade to be assigned to the submission
Returns: Ok value if successful

newPushEvent(owner,name)
Function call to start the automatic evaluation of a submission
Param: owner the username of the owner of the repository
Param: name the name of the repository
Returns: Ok value if successful

2.5.10 Authentication Manager

authenticate(code)
Rest API callback used by github to redirect a user after they completed authentication
Param: code the code needed to create a new access code for the user
Returns: info about the user that has just been authenticated if successful

2.6 Selected architectural Styles and patterns

2.6.1 4-tier Architecture

CKB will be built over a 4-tier architecture which provides many benefits thanks to the modularization of the system in three independent layers (or tiers): Presentation tier: Top-level tier including the user's interface. Its main function is to rearrange the received data from the web and application tiers and present them to the user in a more intuitive and comprehensible manner. Web Server tier: this tier receives requests via HTTP from the user's client and redirects them to an application server that is available. After the web server receives the response from the application tier, it will forward the response message to the user. Moreover, this tier is going to have a caching funcion (it will send static files to the user, when required) and a load balancing function: the requests received are sent to a server that has enough resources to elaborate it. Application tier: It includes the business logic of the application, that is the logic according to which the application takes decisions and performs calculations. Moreover, it passes and processes data between the two surrounding layers. Data tier: It includes the Database system and provides an API, to the application tier, for accessing and managing the data stored in the DB Adopting this type of architecture guarantees a higher flexibility by allowing at first to develop and then to update a specific part of the system apart from the others. Besides, the middle tiers between client and data server ensures a better protection of the stored data. In fact the information is accessed through the application layer instead of being accessed directly by the client.

2.6.2 Model View Controller pattern

For implementing the web application, that allows the user to access all the functionalities offered by CKB, was chosen to follow the Model-View-Controller pattern (MVC pattern). It includes three main components:

- Model: contains the application's data and provides methods for its manipulation
- View: it contains all the different possible visual representations of the data
- Controller: sits between the model and the view. Upon the occurrence of events (for example the pression of a button) execute a predefined reaction which may include some operations on the model that are then reflected on the view

The distribution of tasks between these three elements translates into an increased decoupling of the components which leads to a series of benefits such as reusability, simplicity of implementation, etc.

2.6.3 Facade pattern

The facade pattern is used in the implementation of the Http Manager component to hide the complexity of the application server to the client applications to which is provided a simpler interface. This solution ulteriorly increases the decoupling of the various components of our system; in particular between the client applications and application servers.

2.6.4 Mediator pattern

The mediator pattern is a behavioral pattern used to reduce coupling between modules willing to communicate with each other. The mediator component sits in between two or more objects and encapsulates how such objects communicate; this way not requiring them to know implementation details about each other. In our system there is one mediator:

- GitHub Manager acts as a mediator linking the internal components of the application server with the API provided by GitHub. This solution guarantees a greater flexibility of the system which can adapt to a change of the GitHub's API by simply modifying the MapsManager.

2.7 Other design decisions

2.7.1 Availability

Load balancing and replication are concepts that have been introduced in the system to increase the availability and reliability of the system. In this way the system would be as fault-tolerant as possible regarding data management and service availability.

2.7.2 User Session

The user session is being tracked between http calls thanks to an encrypted JWT in the request cookies. This JWT contains:

- UserID internal id
- Github API user access token
- Github username

the encryption of the JTW is a critical aspect of the system security.

2.7.3 Notifications

New notifications are loaded when the user loads the home page.

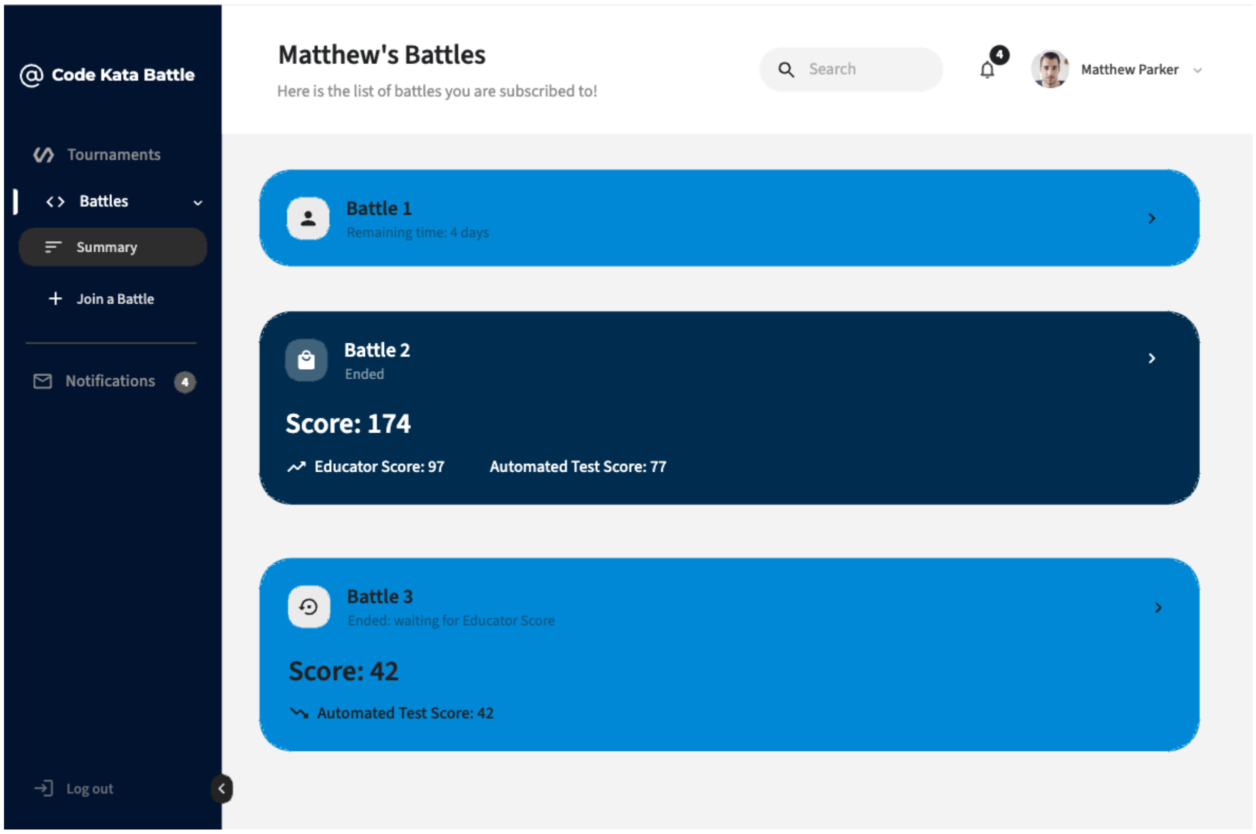
2.7.4 Data Storage

A single SQL database is present in the design. This could introduce availability and performance issues, to compensate the database should be deployed on a separated device allowing it to have the maximum ammound of recources. On the other hand having a single SQL database allows the system data storage to preserve ACID proprierties, which would be impossible with a distributed database. Periodicals backup of the database are needed to increase the system reliability.

2.7.5 Rust backend

The rust backend allows the system to handle high ammounts of requests consuming as little resources as possibles. this allows the system to be very efficent as the majority of the requests are mainly interactions with the database and have little data transformation overhead.

3 User Interface Design



Matthew's Battles

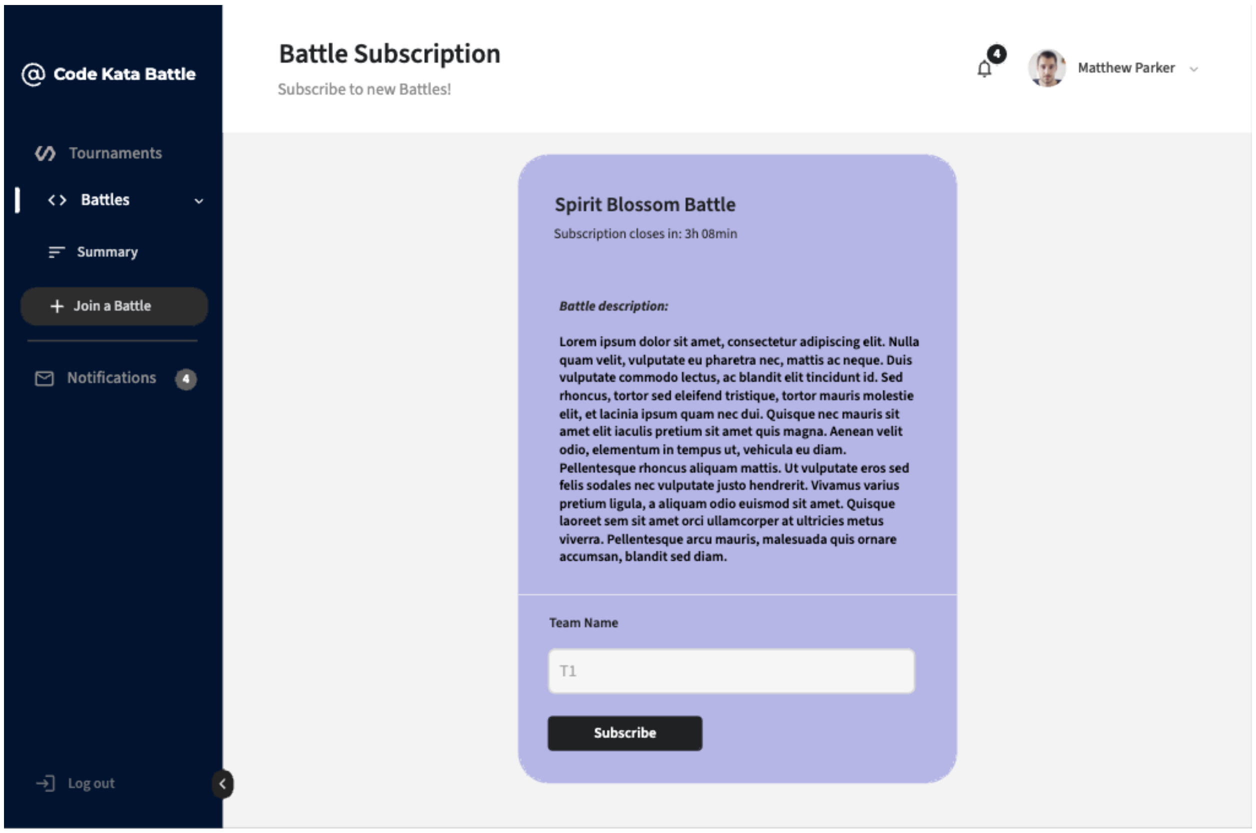
Here is the list of battles you are subscribed to!

Battle 1
Remaining time: 4 days

Battle 2
Ended
Score: 174
Educator Score: 97
Automated Test Score: 77

Battle 3
Ended: waiting for Educator Score
Score: 42
Automated Test Score: 42

Figure 18: Mock up educator's battles view



Battle Subscription

Subscribe to new Battles!

Spirit Blossom Battle

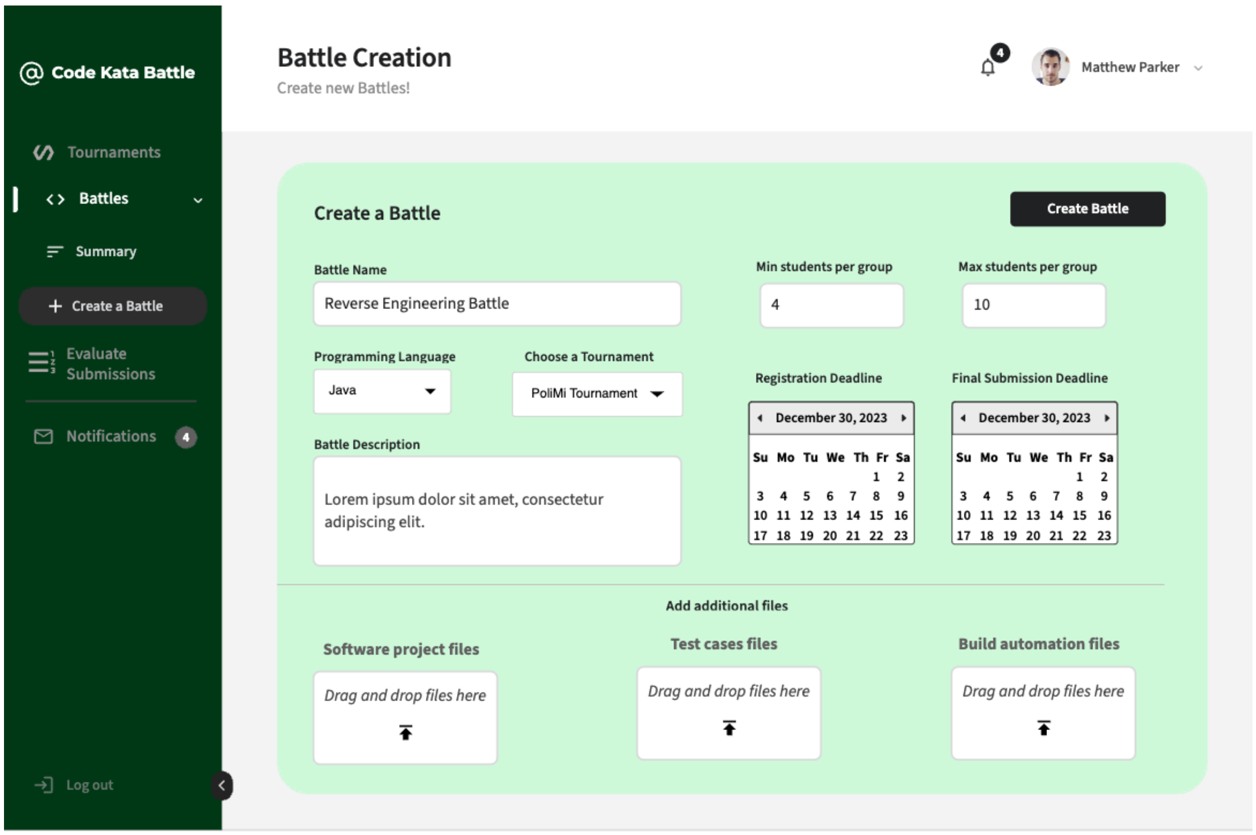
Subscription closes in: 3h 08min

Battle description:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui. Quisque nec mauris sit amet elit iaculis pretium sit amet quis magna. Aenean velit odio, elementum in tempus ut, vehicula eu diam. Pellentesque rhoncus aliquam mattis. Ut vulputate eros sed felis sodales nec vulputate justo hendrerit. Vivamus varius pretium ligula, a aliquam odio euismod sit amet. Quisque laoreet sem sit amet orci ullamcorper at ultricies metus viverra. Pellentesque arcu mauris, malesuada quis ornare accumsan, blandit sed diam.

Team Name: T1

Figure 19: Mock up team creation



Battle Creation

Create new Battles!

Create a Battle

Battle Name: Reverse Engineering Battle

Min students per group: 4

Max students per group: 10

Programming Language: Java

Choose a Tournament: PoliMi Tournament

Registration Deadline: December 30, 2023

Final Submission Deadline: December 30, 2023

Battle Description: Lorem ipsum dolor sit amet, consectetur adipiscing elit.

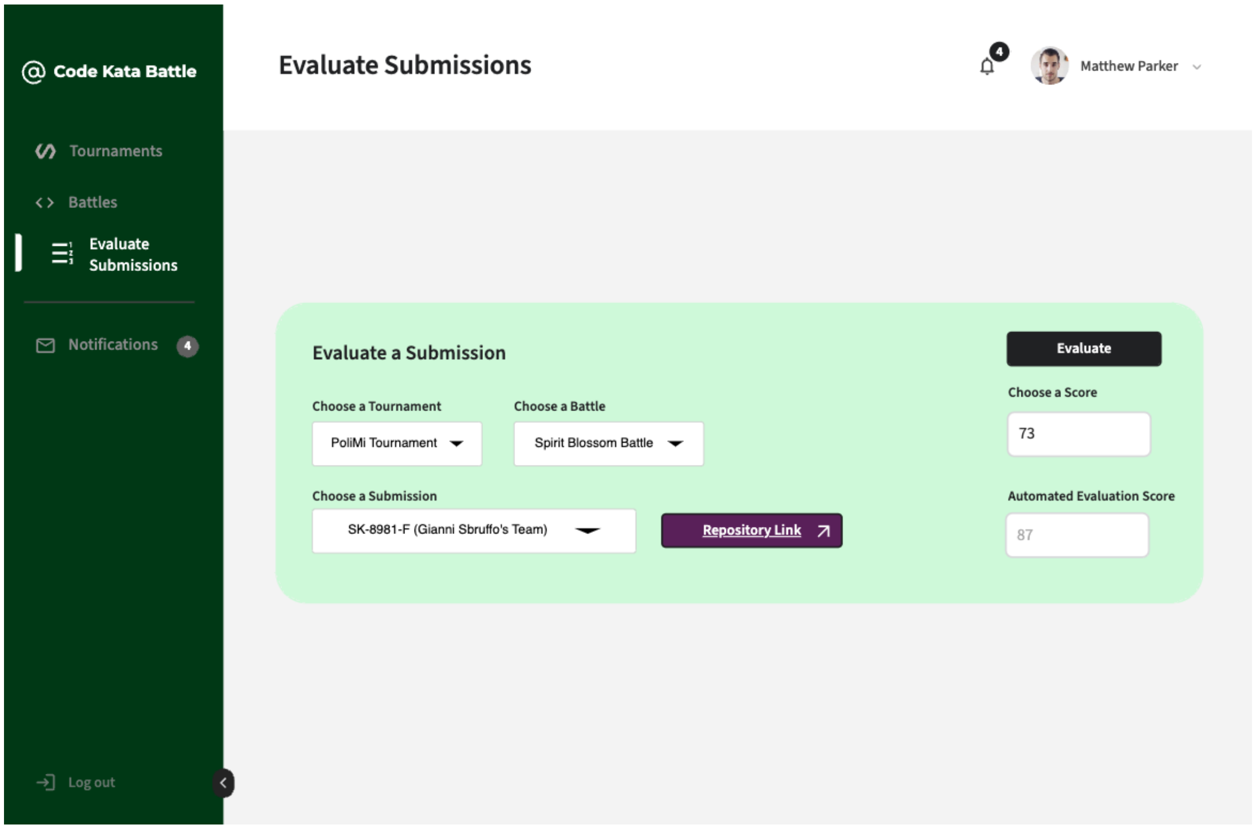
Add additional files

Software project files

Test cases files

Build automation files

Figure 20: Mock up battle creation



Evaluate Submissions

Evaluate a Submission

Choose a Tournament: PoliMi Tournament

Choose a Battle: Spirit Blossom Battle

Choose a Submission: SK-8981-F (Gianni Sbruffo's Team)

Choose a Score: 73

Automated Evaluation Score: 87

Figure 21: Mock up submission evaluation

4 Requirement traceability

4.1 [R1]

The system allows Students to log in via GitHub.

- CodeKata Web App
- Web Server
- Http Manager
- Authentication Manager
- GitHub Manager
- Model
- Database

4.2 [R2]

The system allows Educators to log in via GitHub.

- CodeKata Web App
- Web Server
- Http Manager
- Authentication Manager
- GitHub Manager
- Model
- Database

4.3 [R3]

The system allows Educators to create a tournament.

- CodeKata Web App
- Web Server
- Http Manager
- Tournament Manager
- Model
- Database

4.4 [R4]

The system allows Educators to view a list of only tournaments they manage.

- CodeKata Web App
- Web Server
- Http Manager
- Tournament Manager
- Model
- Database

4.5 [R5]

The system allows Educators to add other Educators to a tournament they manage.

- CodeKata Web App
- Web Server
- Http Manager
- Tournament Manager
- Model
- Database

4.6 [R6]

The system allows Educators to view a list of all Educators that manage the tournament they are managing.

- CodeKata Web App
- Web Server
- Http Manager
- Tournament Manager
- Model
- Database

4.7 [R7]

The system allows Educators to end a tournament they manage.

- CodeKata Web App
- Web Server
- Http Manager
- Tournament Manager
- Model
- Database

4.8 [R8]

The system allows Educators to view a list of Students that are enrolled in a tournament they manage.

- CodeKata Web App
- Web Server
- Http Manager
- Tournament Manager
- Model
- Database

4.9 [R9]

The system allows Educators to create a battle in a tournament they manage.

- CodeKata Web App
- Web Server
- Http Manager
- Tournament Manager
- Battle Manager
- Model
- Database

4.10 [R10]

The system allows Educators to view a list of teams that are participating to a battle they have created.

- CodeKata Web App
- Web Server
- Http Manager
- Battle Manager
- Model
- Database

4.11 [R11]

The system allows Educators to view a list of submitted solutions for a battle they have created.

- CodeKata Web App
- Web Server
- Http Manager
- Authentication Manager
- GitHub Manager
- Battle Manager
- Model
- Database

4.12 [R12]

The system allows Educators to view the repository of each solution submitted to a battle they have created.

- CodeKata Web App
- Web Server
- Http Manager
- Authentication Manager
- GitHub Manager
- Battle Manager
- Model
- Database

4.13 [R13]

The system allows Educators to grade a solution for a battle they have created, after the deadline of a battle.

- CodeKata Web App
- Web Server
- Http Manager
- Battle Manager
- Submission Evaluation Manager
- Model
- Database

4.14 [R14]

The system allows Educators to see the grade given by the automated tests to a solution of a battle.

- CodeKata Web App
- Web Server
- Http Manager
- Battle Manager
- Submission Evaluation Manager
- Model
- Database

4.15 [R15]

The system allows Educators to view the rank scoreboard of a tournament they manage.

- CodeKata Web App
- Web Server
- Http Manager
- Battle Manager
- Submission Evaluation Manager
- Model
- Database

4.16 [R16]

The system allows Students to view a list of all available tournaments.

- CodeKata Web App
- Web Server
- Http Manager
- Tournament Manager
- Model
- Database

4.17 [R17]

The system allows Students to join a tournament.

- CodeKata Web App
- Web Server
- Http Manager
- Tournament Manager
- Model
- Database

4.18 [R18]

The system allows Students to view a list of tournaments they are enrolled in.

- CodeKata Web App
- Web Server
- Http Manager
- Tournament Manager
- Model
- Database

4.19 [R19]

The system allows Students to view a list of all available battles of a given tournament.

- CodeKata Web App
- Web Server
- Http Manager
- Tournament Manager
- Battle Manager
- Model
- Database

4.20 [R20]

The system allows Students to create a team for a battle.

- CodeKata Web App
- Web Server
- Http Manager
- Battle Manager
- Model
- Database

4.21 [R21]

The system allows Team Leaders to invite other students to their team.

- CodeKata Web App
- Web Server
- Http Manager
- Battle Manager
- Model
- Database

4.22 [R22]

The system allows Students to view a list of invitations received.

- CodeKata Web App
- Web Server
- Http Manager
- Battle manager
- Model
- Database

4.23 [R23]

The system allows Students to accept an invite.

- CodeKata Web App
- Web Server
- Http Manager
- Battle manager
- Model
- Database

4.24 [R24]

The system allows Students to decline an invite.

- CodeKata Web App

4.25 [R25]

The system allows Students to view a list of battles they have joined, given a tournament.

- CodeKata Web App
- Web Server
- Http Manager
- Tournament manager
- Model
- Database

4.26 [R26]

The system allows Students to view the description of a battle they are enrolled into.

- CodeKata Web App
- Web Server
- Http Manager
- Battle manager
- Model
- Database

4.27 [R27]

The system allows Students to view the repository of a battle they are enrolled into.

- CodeKata Web App
- Web Server
- Http Manager
- Battle manager
- Model
- Database

4.28 [R28]

The system allows Students to view the score of the solution submitted, given a battle, and after the solution has been graded.

- CodeKata Web App
- Web Server
- Http Manager
- Battle manager
- Model
- Database

4.29 [R29]

The system allows Students to view the rank scoreboard of a tournament they are enrolled in.

- CodeKata Web App
- Web Server
- Http Manager
- Tournament manager
- Model
- Database

4.30 [R30]

The system runs GitHub Actions on battles' repository.

- Github Manager

4.31 [R31]

The system detects when a new version of the main branch is available.

- CodeKata Web App
- Web Server
- Http Manager

4.32 [R32]

System notifies student they have been invited to a team.

- CodeKata Web App
- Web Server
- Http Manager
- Notification manager
- Model
- Database

4.33 [R33]

System assign automatic grading.

- CodeKata Web App
- Web Server
- Http Manager
- Submission Evaluation manager
- Github Manager
- Model
- Database

4.34 [R34]

System notifies students a new battle is available.

- CodeKata Web App
- Web Server
- Http Manager
- Notification manager
- Model
- Database

4.35 [R35]

System notifies students a new tournament is available.

- CodeKata Web App
- Web Server
- Http Manager
- Notification manager
- Model
- Database

4.36 [R36]

System notifies a tournament has been closed.

- CodeKata Web App
- Web Server
- Http Manager
- Notification manager
- Model
- Database

4.37 [R37]

System notifies a student that a battle has ended.

- CodeKata Web App
- Web Server
- Http Manager
- Notification manager
- Model
- Database

4.38 [R38]

System notifies a team their solution has been graded.

- CodeKata Web App
- Web Server
- Http Manager
- Notification manager
- Model
- Database

4.39 [R39]

System notifies educators that a new submission is ready to be graded.

- CodeKata Web App
- Web Server
- Http Manager
- Notification manager
- Model
- Database

4.40 [R40]

System notifies students that the student they have invited to their team has accepted/rejected the invite.

- CodeKata Web App
- Web Server
- Http Manager
- Notification manager
- Model
- Database

4.41 [R41]

System is able to fork the educator's repository.

- Github manager
- Model
- Database

5 Implementation, Integration and Test Plan

5.1 Overview

This chapter describes how the implementation and the system will be executed. The order with which the components will be implemented and tested. The kind test that will be executed on the system as integration tests and as system tests.

5.2 Implementation plan

The implementation will follow a bottom-up approach to reduce the amount of stubs needed for integration testing, although stub and drivers will still be needed for unit testing. After the implementation and the modification of a module unit tests will be run. Each component will expose the interfaces listed in the 2.5 subsection.

Components will be divided in ranks, components in the same ranks can be implemented in parallel as they are independent from one another. The system will be implemented from the first to the latest rank, one rank at the time.

5.2.1 First rank: Database

The database's schema design will be the first step of the implementation. For every end point listed on the 2.5 subsection a corresponding prepared statement will be implemented, this will improve the database performance and will make query to the database for the needed data easter and independent from the true structure of the database schema.

5.2.2 Second rank: Model and Github Manager

The Model will require a test database in order to execute unit testing.

The Github Manager will require a stub for the unit, integration and system testing as the github api is rate limited and using the real api for testing will reduce the ammount of request available to the system.

5.2.3 Third rank: Tournament, Authentication, Battle, Submission evaluation and Notification Manager

These five components are indipenedt from each other and can be developed in parallel.

5.2.4 Fourth rank: Http Manager

The Http Manager will expose an http server. The unit and integration test's drivers will be required to connect to the http server.

5.2.5 Fifth rank: Web Server

The Web server's configuration will define the load balancing system and the static files distribution. As the web server is an external unit testing will be only responsable to assert the correct configuration in order to more easely identify error in it without having to wait until the integration testing.

5.2.6 Sixth rank: CodeKata WebApp

The Web app implementation will be the final step. The unit testing of the webapp will require a dedicated framework to automate the interface testing.

5.3 Component Integration and Testing

Each component will have a driver and stud used for the unit testing of that component. Regaridng integration testing, system integration will follow the implementation order and outised than the Model and Github Manager, components will not need studs. For each step all the top component will have their own driver that will execute the testing in parallel with the other drivers. The model stud will be copy of the production database schema.

Here is a list of images describing the steps of the integration testing process:

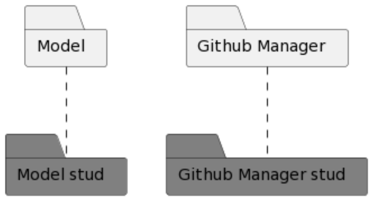


Figure 22: First step of integration testing

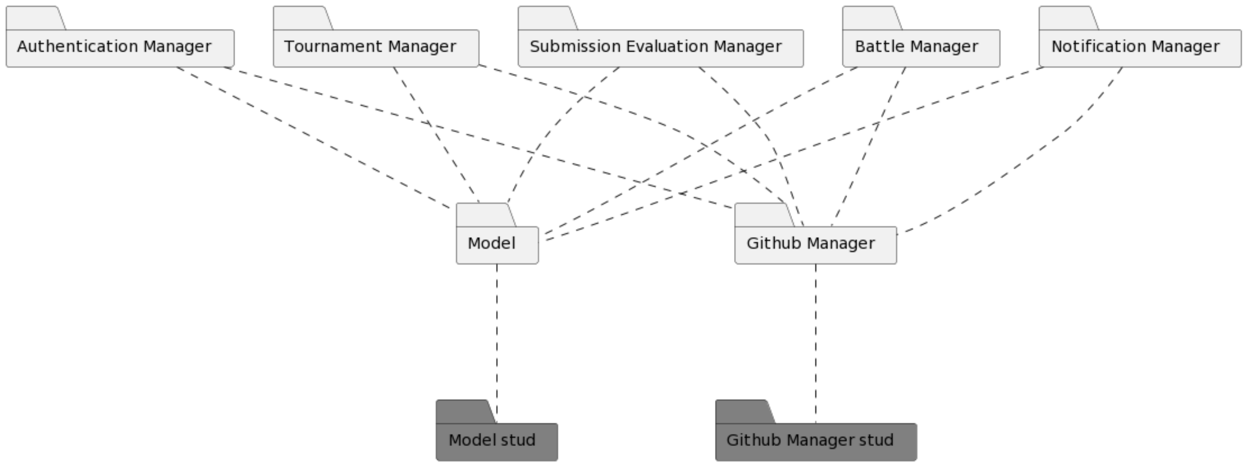


Figure 23: Second step of integration testing

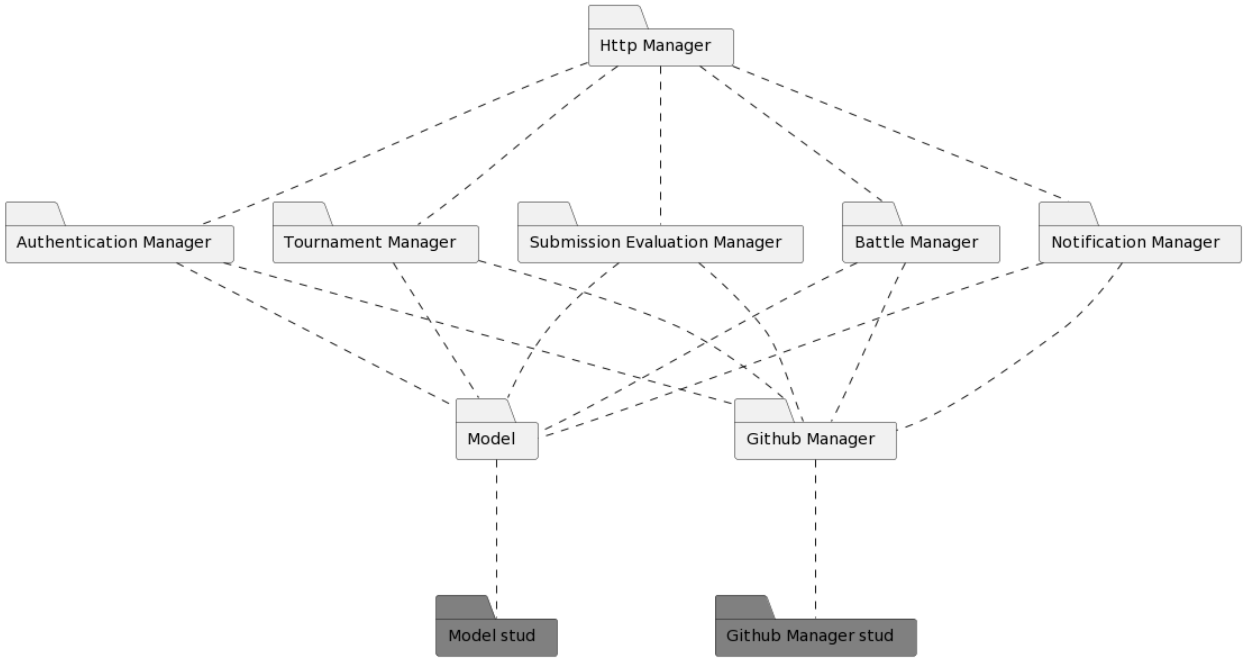


Figure 24: Third step of integration testing

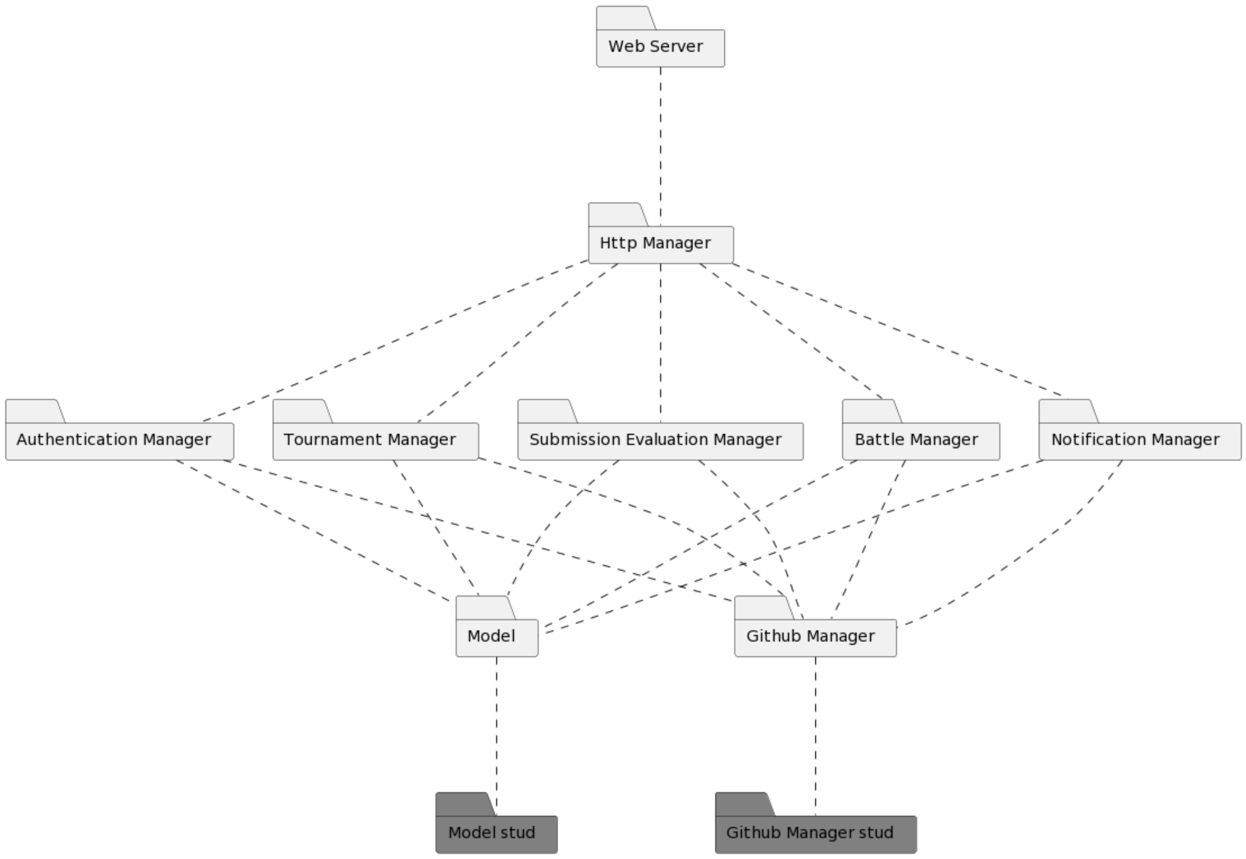


Figure 25: Fourth step of integration testing

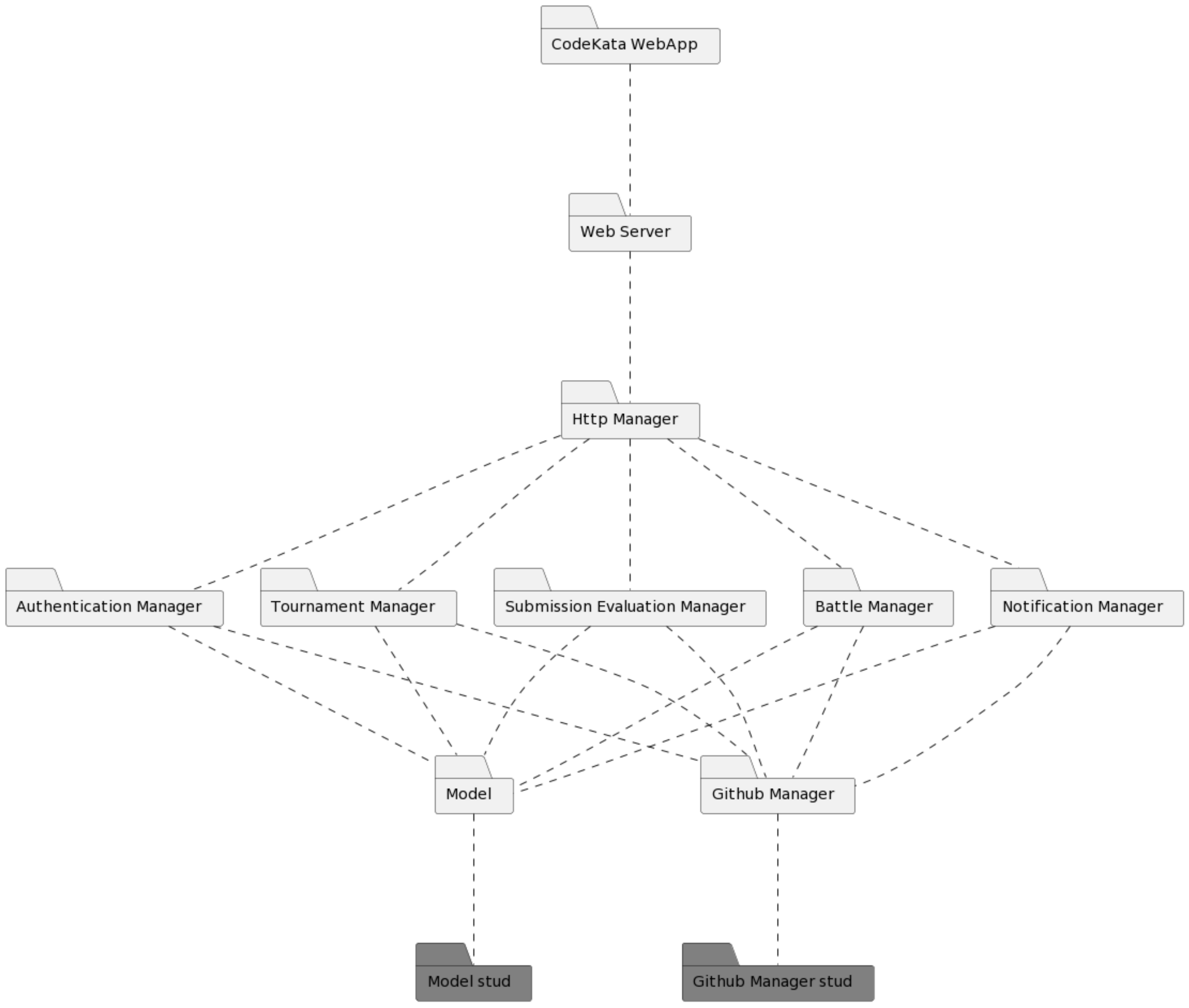


Figure 26: Fifth step of integration testing

5.4 System testing

After creating the components, unit testing them, integrating the system and testing the system integration the system will be testend in it's entierty to verify that all functions have been developed correctly and the system comply with the reqirements. Different aspect of the system will be tested such as:

- Functional testing: this will verify that the system satisfies the functional requirements defined in the RASD.
- Performance testing: this will verify that the system satisfies the performance requirements and it will help identify the system real performances make it able to find system inefficecy if they are present.
- Usability testing: this will verify how easy is to use the system and if there are problem with the user experience.
- Load testing: this will verify how the system behaves under heavy load and will help find resource wasting bugs if they are present.
- Stress testing: this will verify how the system recovers after a failure.

5.5 Additional specification on testing

All the test will need to be repeated every time some part of the implementation is changed, this is done to maximise the system relaiability.

6 Effort Spent

Merlino Lorenzo	2h	22h	2h	4h	5h
Iodice Andrea	2h	12h	6h	3h	2h

7 References

- UML diagrams: plantUML, starUML