CodeKataBattle

Software Engineering 2 Project - DD v1.0

Mariotti Marco
Personal code: 10939419

7 January 2024

Contents

1 Introduction

1.1 Purpose

This document contains the description of the architectural design of the *CodeKataBattle* application in alignment with the specifications outlined in the *RASD document.* The document specifies essential design characteristics by introducing constraints and quality attributes required by the implementation. It also provides a comprehensive overview of the implementation plan, integration plan, and testing plan.

1.2 Scope

CodeKataBattle will be built upon a **3-tier client server architecture**. In order to guarantee availability and address performance issues, the Application Tier containing the Web Server and the Data Tier containing the Database Server will be replicated. The CKB will be accessible through web browsers on both desktop and mobile devices for both students and educators; the user interface will be designed with distinct implementations tailored to each device type. Considering the diverse user roles, interface implementations will provide different functionalities, addressing the specific needs of each user interacting with the system.

The system also interacts with third party applications, in particular a GitHub API will be used to manage the repositories containing the various CodeKata, and the evaluation system will rely on static analysis tools (an example: SonarQube).

1.3 Definitions, Acronyms, Abbreviations

Definitions:

- **CodeKataBattle:** The software to be developed.
- **Student:** The participant in one or more tournaments and battles
- **Educator:** The creator of one or more tournaments and battles
- **Tournament:** A competition composed of multiple battles
- **Battle:** A contest of multiple teams to create a project that satisfies all the tests

Acronyms:

- **CKB:** CodeKataBattle

1.4 Document structure

This document is composed of the chapters described below.

- **Chapter 1** describes the scope and purpose of the DD, including the structure of the document and the set of definitions, acronyms and abbreviations used.
- **Chapter 2** contains the architectural design choice, it includes all the components, the interfaces, the technologies (both hardware and software) used for the development of the application. It also includes the main functions of the interfaces and the processes in which they are utilised (runtime view and component interfaces). Finally, there is the explanation of the architectural patterns chosen with the other design decisions.
- **Chapter 3** The third chapter contains mockups for the user interface of the system to be.
- **Chapter 4** describes the connection between the RASD and the DD, showing the matching between the goals and requirements described previously with the elements which compose the architecture of the application.
- **Chapter 5** traces a plan for the development of components to maximize the efficiency of the developer team and the quality controls team. It is divided in two sections: implementation and integration. It also includes the testing strategy.
- **Chapter 6** shows the effort spent for each member of the group.
- **Chapter 7** lists the eventual references used in this document.

2 Architectural Design

2.1 Overview

The CKB application will rely on a 3-Tier Architecture. This type of architecture is well established, and because of the fact that each tier runs on its own infrastructure, each tier can be deveoped and mantained simultaneously, wihout impacting other tiers. The three tiers are the following:

- *Presentation*: this tier incorporates the user interface and communication layer of the application, managing the interaction of the end user with the application itself. It is the top-level tier of the architecture, it can run on a desktop application, a web browser (like in the case of CKB) or on any graphical user interface.
- *Application*: the application tier, also called logic tier, incorporates the buisiness logic behind the application. The application tier interacts both with the presentation tier and the data tier, overseeing the transfer of data between the other two layers.
- *Data*: the data tier contains the database of the application, so all the information that is stored and processed in the application. It can be implemented by relational database management systems (DBMS) such as PostgreSQL or MySQL.

The system will also interact with some external services, through appropriate APIs. In particular, a mail server for sending notifications to users, GitHub for the project workflow management and any kind of appliaction that provides static analysis tools for code analysis, such as SonarQube.

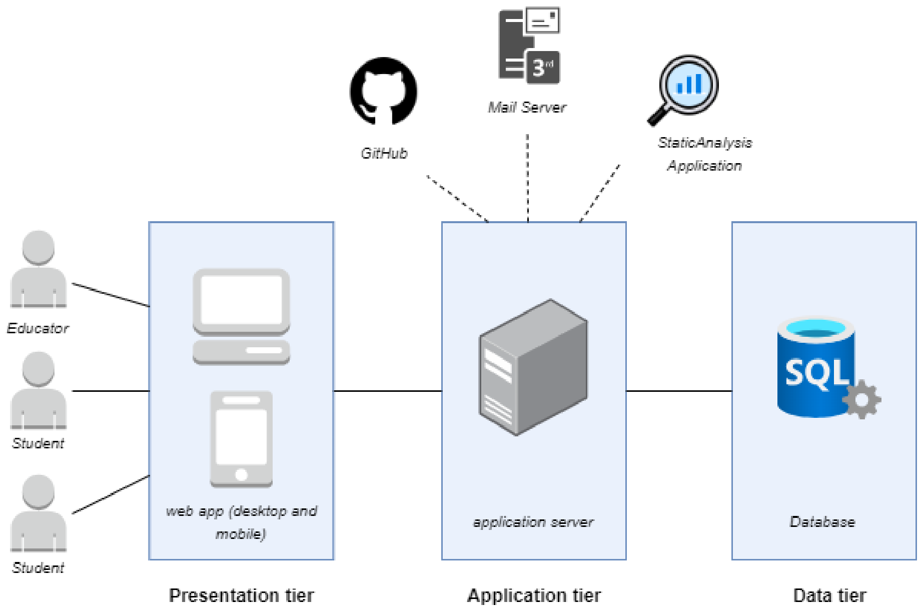


Figure 1: High-level overview of the 3-tier architecture for CKB

2.2 Component View

In the following section the component diagram is included in order show the main components of the system and the relationships between them.

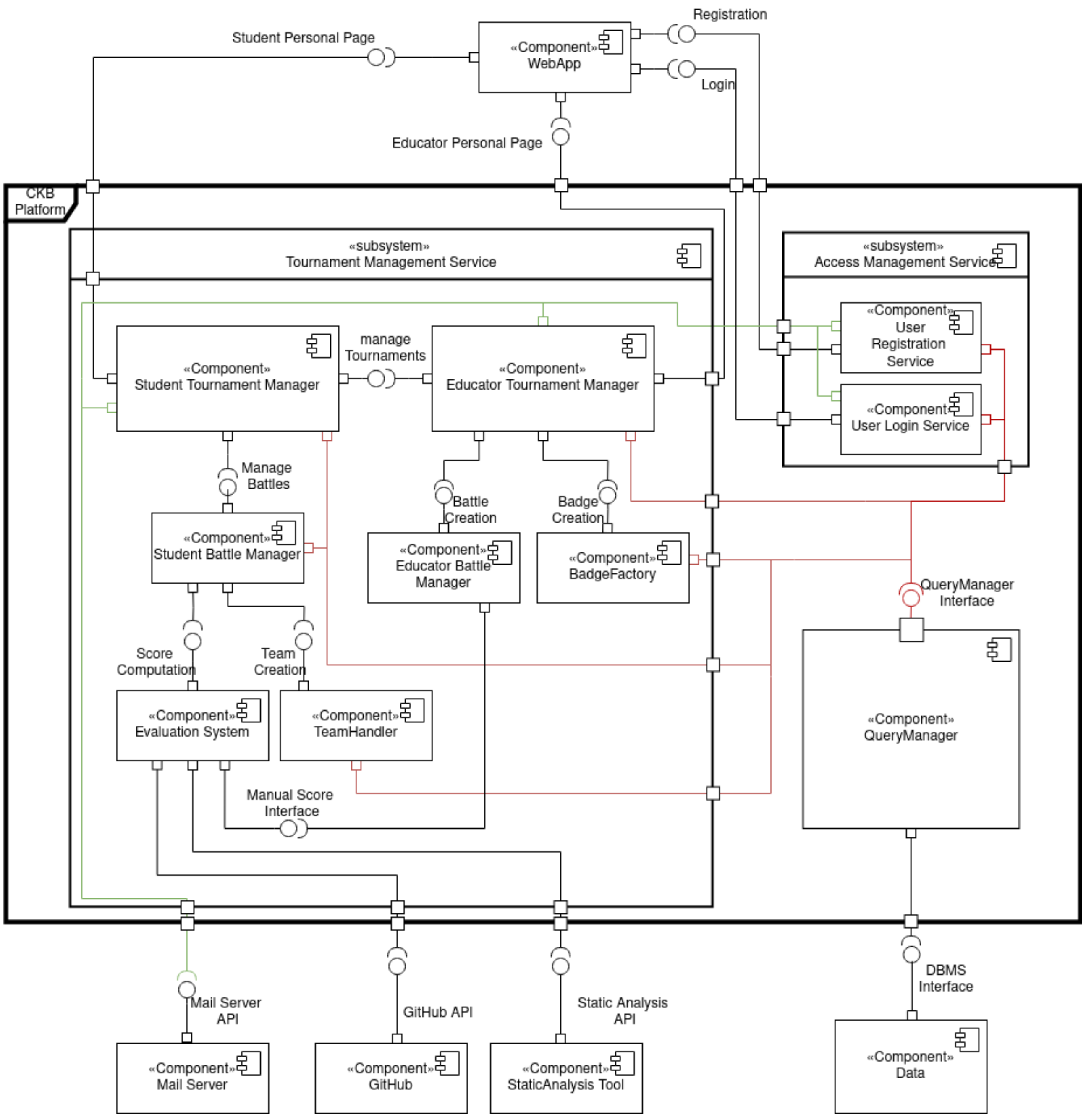


Figure 2: CKB Platform Component Diagram

The CKB Platform consists of the following components (divided with respect to the 3 tiers architecture):

- **Presentation**:
  - `WebApp`: Web browser client application
- **Application**:
  - **Tournament Management Service**: subsystem dedicated to Tournament Management

- ∗ `Educator Tournament Manager`: component containing all the functions allowing an Educator user to create and manage a Tournament
    - ∗ `Educator Battle Manager`: component that contains all the functions allowing an Educator user to create and manage a Battle
    - ∗ `Student Tournament Manager`: component containing all the functions allowing a Student user to browse and join Tournaments
    - ∗ `Student Battle Manager`: component containing all the functions allowing a Student user to join a Battle and actively monitor its progress
    - ∗ `Evaluation System:`: component that provides the score for a Battle
    - ∗ `TeamHandler`: component that handles the Team creation process for a Battle
    - ∗ `BadgeFactory`: component that allows an Educator user to create Badges
  - **Access Manager:** subsystem dedicated to user management
    - ∗ `User Registration Service`: component that handles the registration process to CKB
    - ∗ `User Login Service`: component that allows the login process to CKB
  - **QueryManager**: component that interacts with the database. It collects all the functions containing a query to the DB
- **Data**:
  - **DBMS:** database server
- **External**:
  - `GitHub`: external component that allows the CKB Platform to interact with the repositories of the various Battles.
  - `Static Analysis Tool`: external component that provide the static analysis measurements in order to compute a Battle's score
  - `Mail server`: external component that allows the CKB Platform to send emails to the user

2.3 Deployment View

In this section the deployment diagram is included (figure3) in order to capture the topology of the system's hardware.

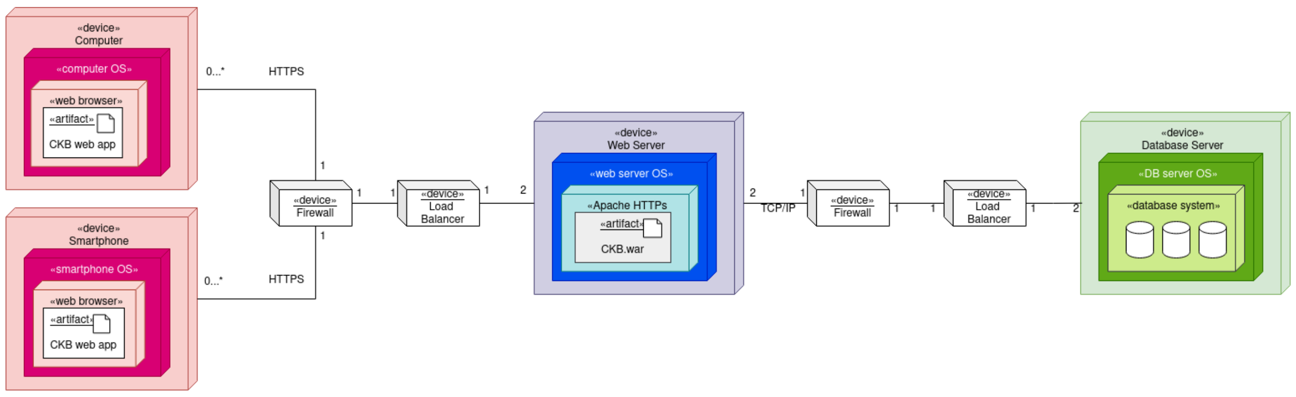


Figure 3: CKB Platform Deployment Diagram

As previously specified, the system implements a 3-tier architecture. In the following, details about each tier is included.

1. **Presentation Tier**: contains the UI part of the system. Users access the CKB Platform throught the web browser installed on their devices (both Computers and Smartphones). An HTTPS connection is needed in order to communicate with the Web Server.
2. **Application Tier**: the *Web Server* containing all the business logic of the CKB Platform is deployed within this tier. In order to guarantee the system's availability and performance, 2 replicas of the web server are deployed, and the distribution of requests coming from the presentation tier is handled by the *Load Balancer*. Additionally, a *Firewall* is in place to monitor both incoming and outgoing traffic. The Application Tier handles requests from the Presentation Tier using the HTTPs protocol, while communicates with the Data Tier through TCP/IP.
3. **Data Tier**: the *Database Server* is deployed within this tier. All data regarding users, statistics, ongoing and past tournament and battles is contained here. The Database Server is accessed by the Application Tier using a TCP/IP connection. Incoming and outgoing traffic is monitored by a *Firewall* and the Database Server is replicated 2 times; the distribution of requests is handled by a *Load Balancer* as well.

2.4 Runtime View

In this section we include sequence diagrams to represent run-time interactions among components defined in the component diagram of section 2.2. Interctions and scenarios will follow those described in Use Cases presented in the RASD document, going into further detail about how the system behaves in relation to interactions among components.

For simpicity's sake and readability, we assume that all data manually inserted by the user is correct, and that errors caused by loss of connection or system failure don't happen. In those cases, operations currently in process are not completed, an error message is displayed and the user is asked to try again.

Student registration

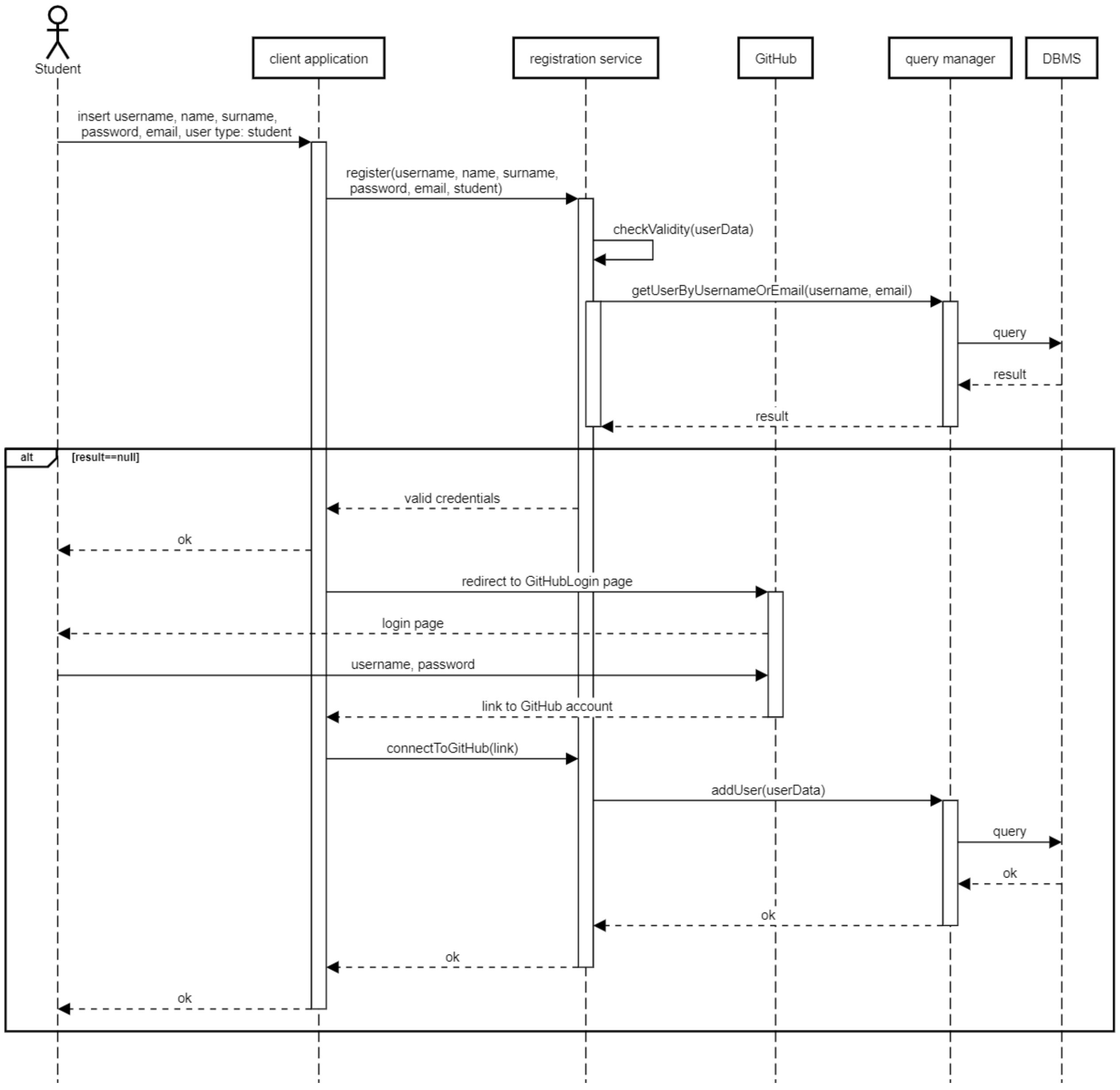


Figure 4: User registers as a student

Educator registration

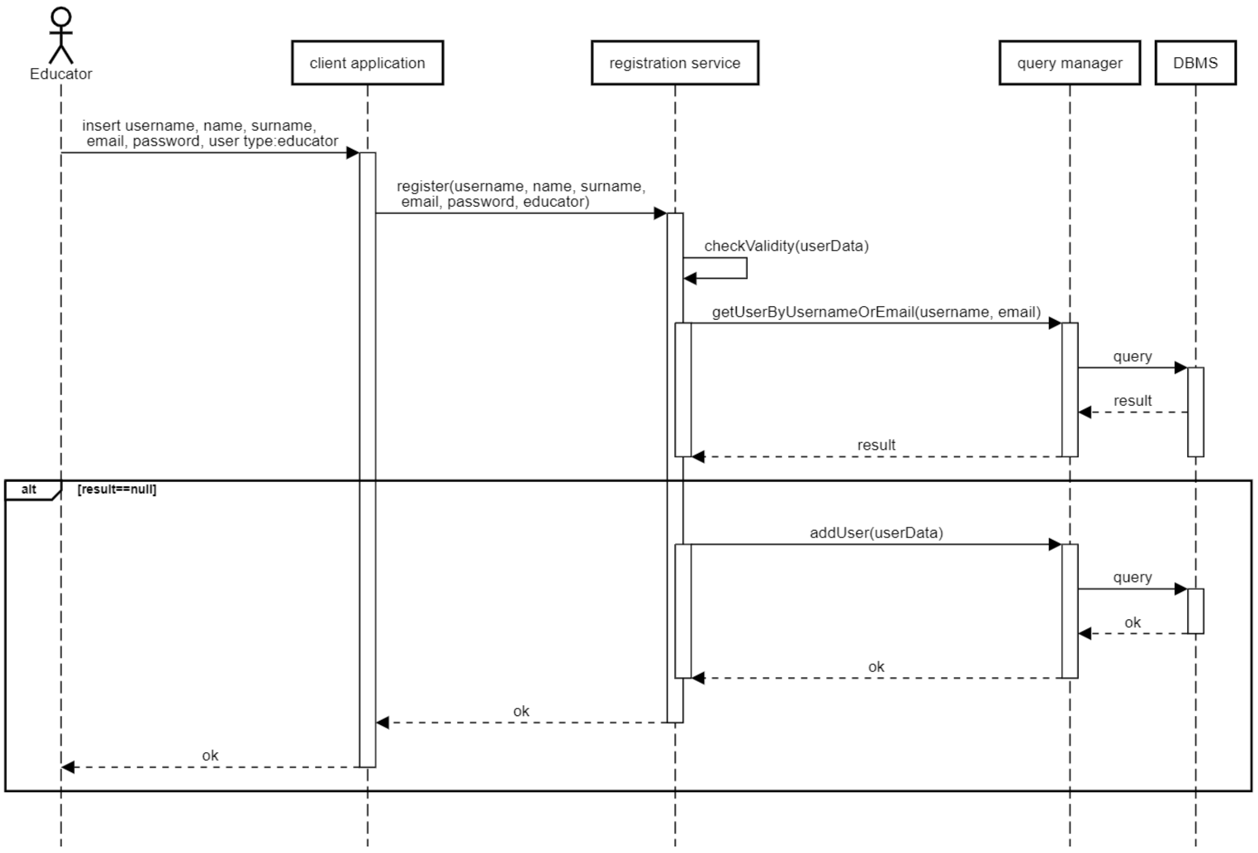

Educator
client application
registration service
query manager
DBMS
insert username, name, surname, email, password, user type:educator
register(username, name, surname, email, password, educator)
checkValidity(userData)
getUserByUsernameOrEmail(username, email)
query
result
result
alt
[result==null]
addUser(userData)
query
ok
ok
ok
ok

Figure 5: User registers as an educator

User login

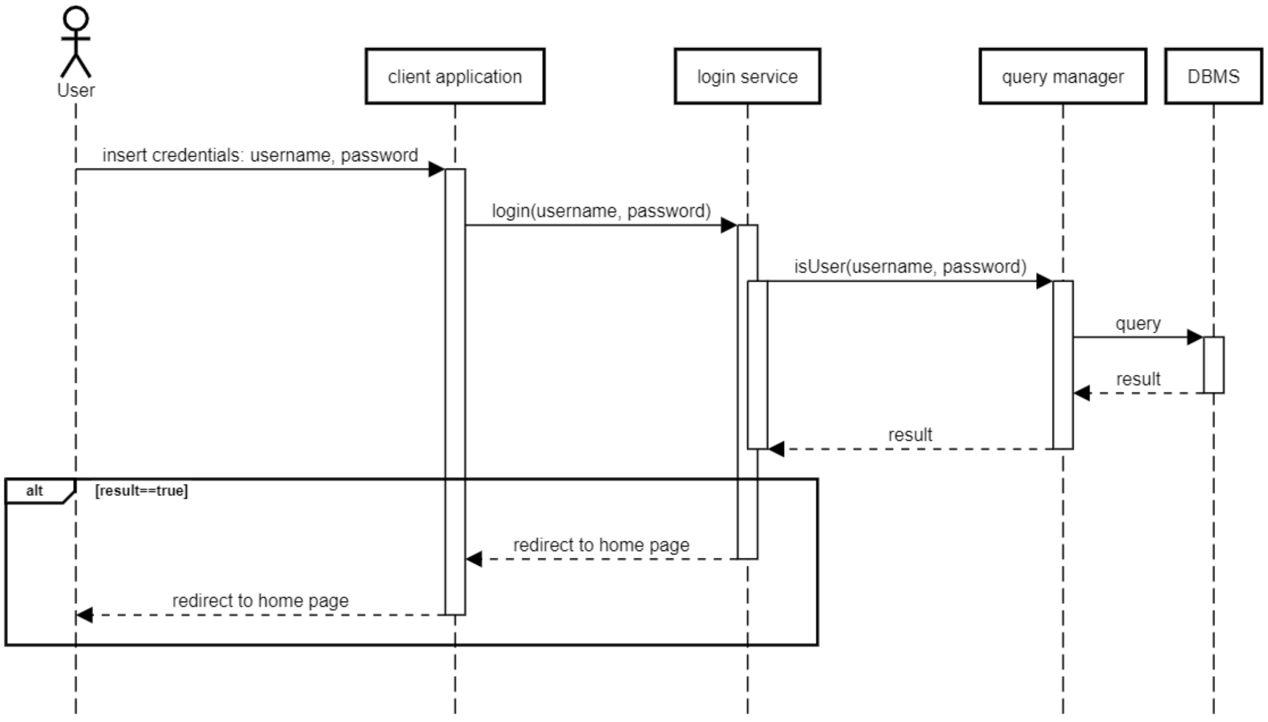


Figure 6: User logs in CKB

Join tournament

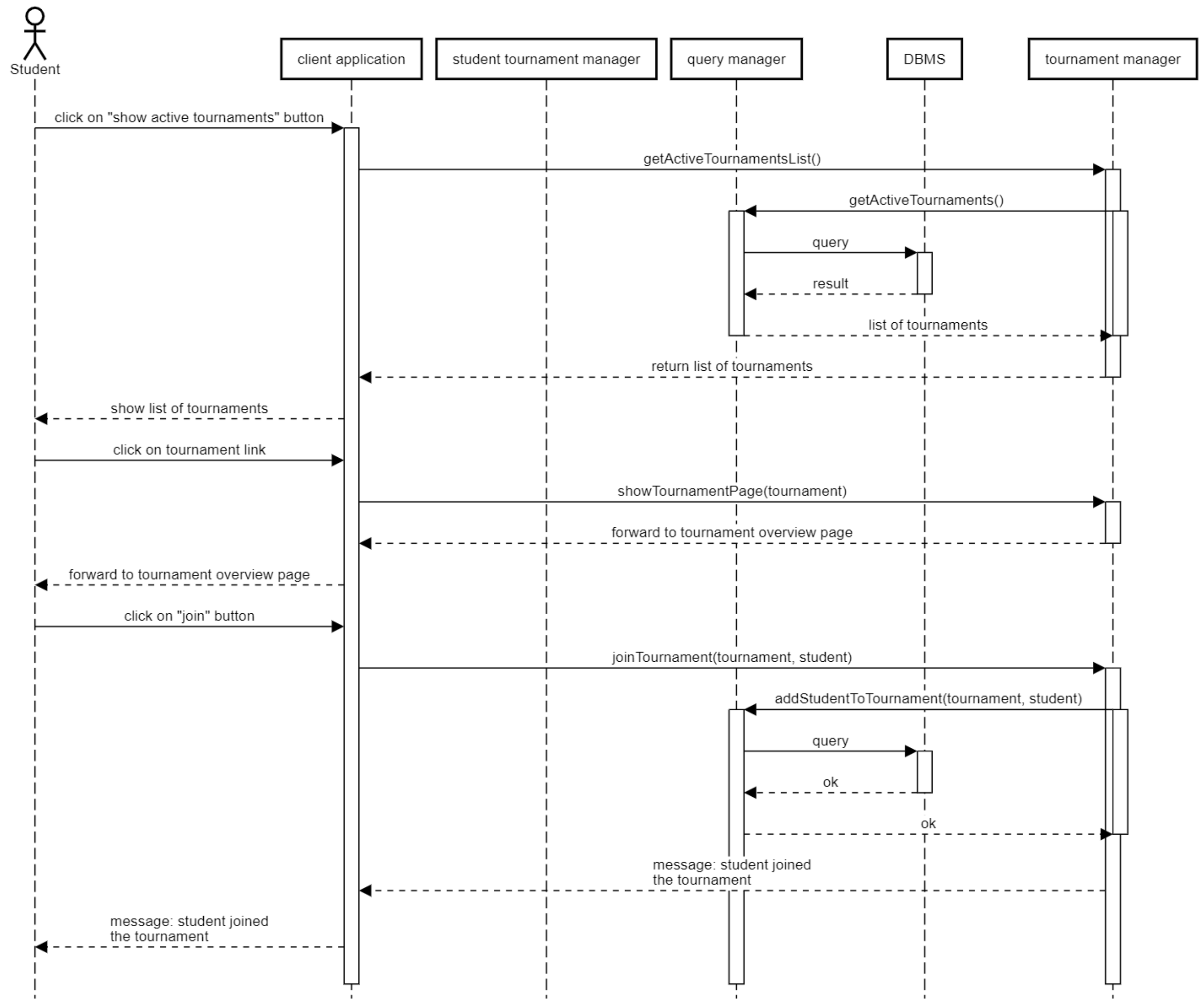

Student
client application
student tournament manager
query manager
DBMS
tournament manager
click on "show active tournaments" button
getActiveTournamentsList()
getActiveTournaments()
query
result
list of tournaments
return list of tournaments
show list of tournaments
click on tournament link
showTournamentPage(tournament)
forward to tournament overview page
forward to tournament overview page
click on "join" button
joinTournament(tournament, student)
addStudentToTournament(tournament, student)
query
ok
ok
message: student joined the tournament
message: student joined the tournament

Figure 7: Student joins a tournament

Create a team

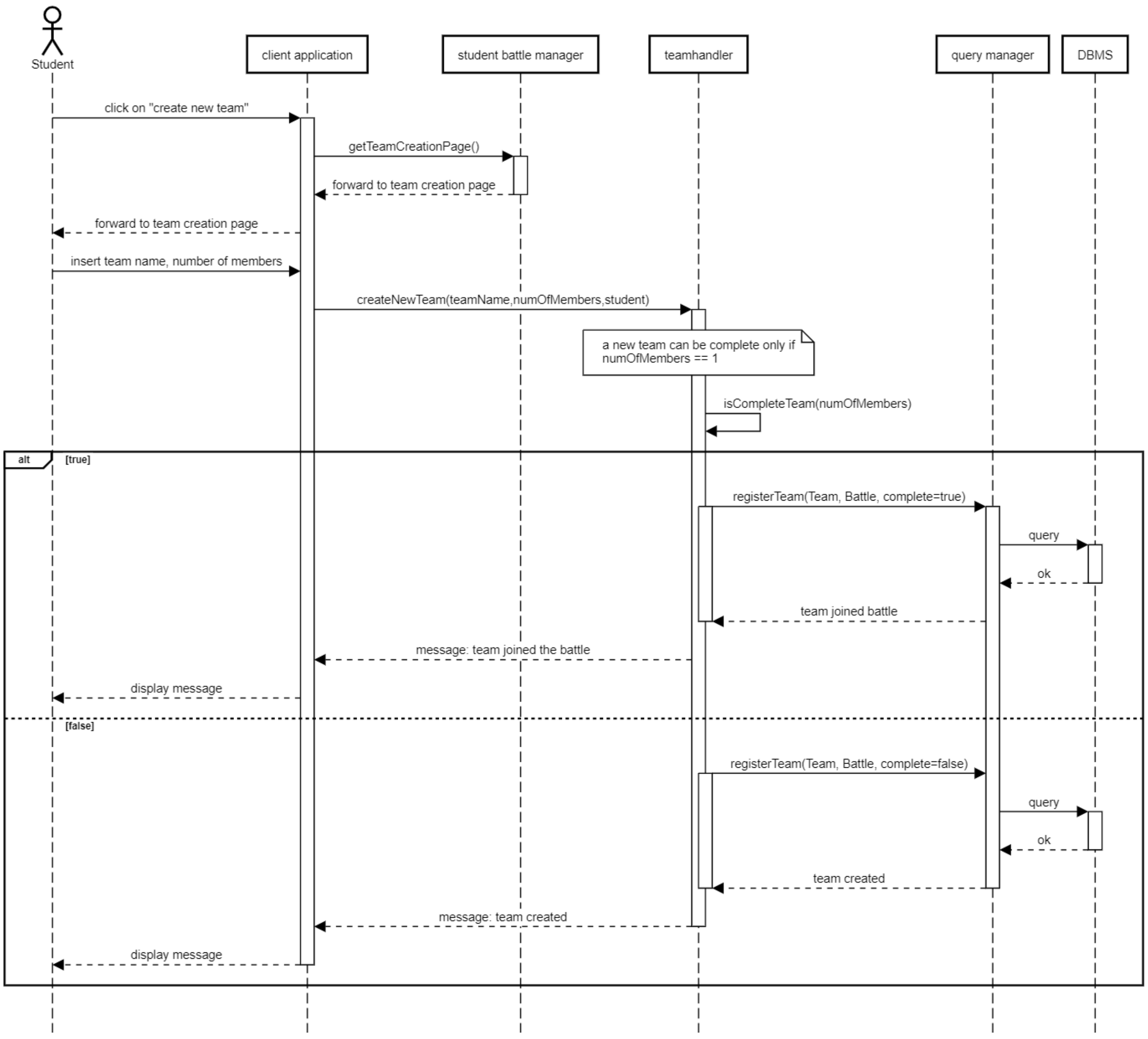

Student
client application
student battle manager
teamhandler
query manager
DBMS
click on "create new team"
getTeamCreationPage()
forward to team creation page
forward to team creation page
insert team name, number of members
createNewTeam(teamName,numOfMembers,student)
a new team can be complete only if numOfMembers == 1
isCompleteTeam(numOfMembers)
alt
[true]
registerTeam(Team, Battle, complete=true)
query
ok
team joined battle
message: team joined the battle
display message
[false]
registerTeam(Team, Battle, complete=false)
query
ok
team created
message: team created
display message

Figure 8: Student creates a new team

Request to join a team

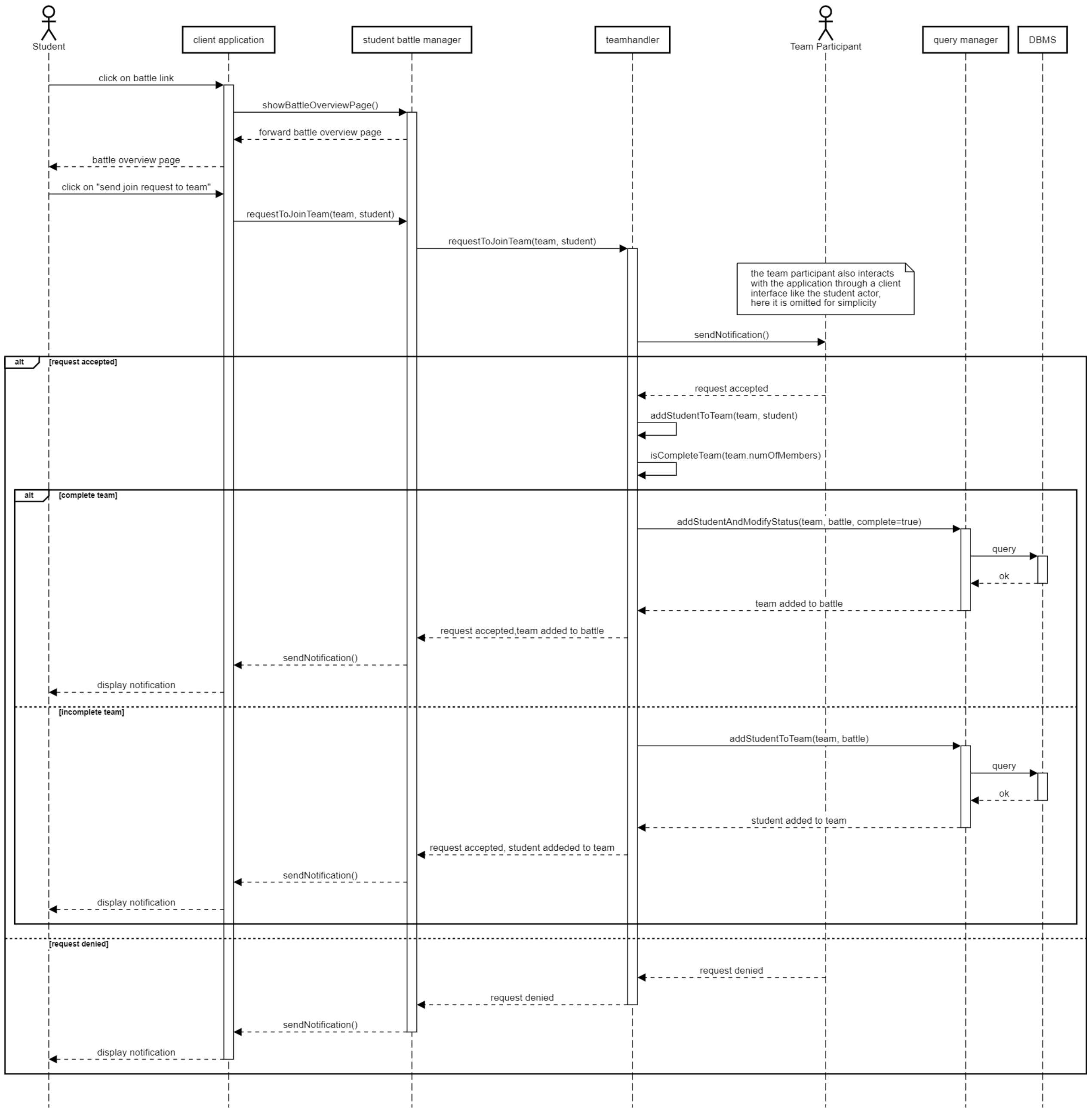

Student
client application
student battle manager
teamhandler
Team Participant
query manager
DBMS
click on battle link
showBattleOverviewPage()
forward battle overview page
battle overview page
click on "send join request to team"
requestToJoinTeam(team, student)
requestToJoinTeam(team, student)
the team participant also interacts with the application through a client interface like the student actor, here it is omitted for simplicity
sendNotification()
alt
[request accepted]
request accepted
addStudentToTeam(team, student)
isCompleteTeam(team.numOfMembers)
alt
[complete team]
addStudentAndModifyStatus(team, battle, complete=true)
query
ok
team added to battle
request accepted,team added to battle
sendNotification()
display notification
[incomplete team]
addStudentToTeam(team, battle)
query
ok
student added to team
request accepted, student addeded to team
sendNotification()
display notification
[request denied]
request denied
request denied
sendNotification()
display notification

Figure 9: A student sends a join request to a team

Invite to join a team

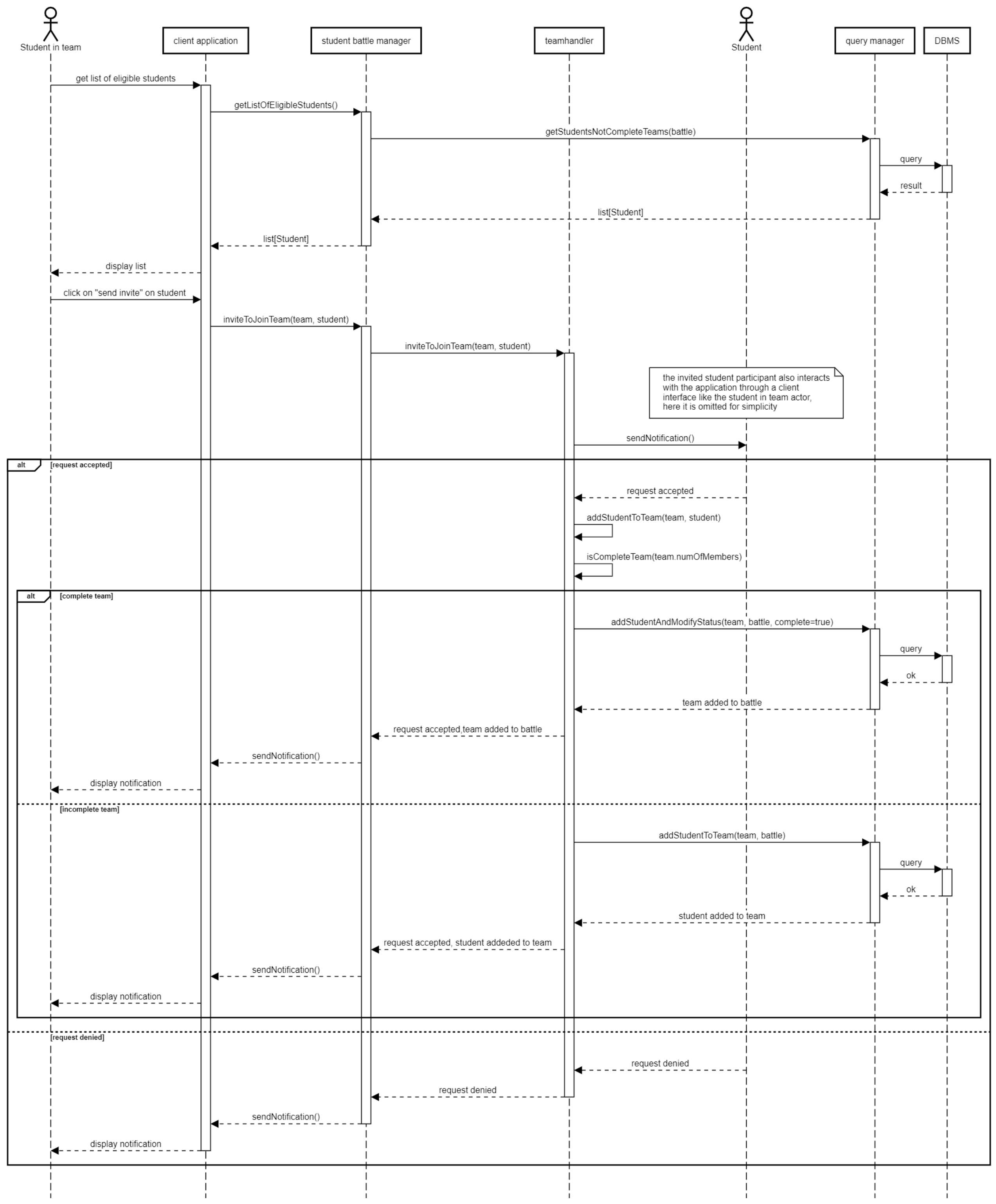

Student in team
client application
student battle manager
teamhandler
Student
query manager
DBMS
get list of eligible students
getListOfEligibleStudents()
getStudentsNotCompleteTeams(battle)
query
result
list[Student]
list[Student]
display list
click on "send invite" on student
inviteToJoinTeam(team, student)
inviteToJoinTeam(team, student)
the invited student participant also interacts with the application through a client interface like the student in team actor, here it is omitted for simplicity
sendNotification()
alt
[request accepted]
request accepted
addStudentToTeam(team, student)
isCompleteTeam(team.numOfMembers)
alt
[complete team]
addStudentAndModifyStatus(team, battle, complete=true)
query
ok
team added to battle
request accepted,team added to battle
sendNotification()
display notification
[incomplete team]
addStudentToTeam(team, battle)
query
ok
student added to team
request accepted, student addeded to team
sendNotification()
display notification
[request denied]
request denied
request denied
sendNotification()
display notification

Figure 10: A student (team member) invites another student to join their team

Quit team

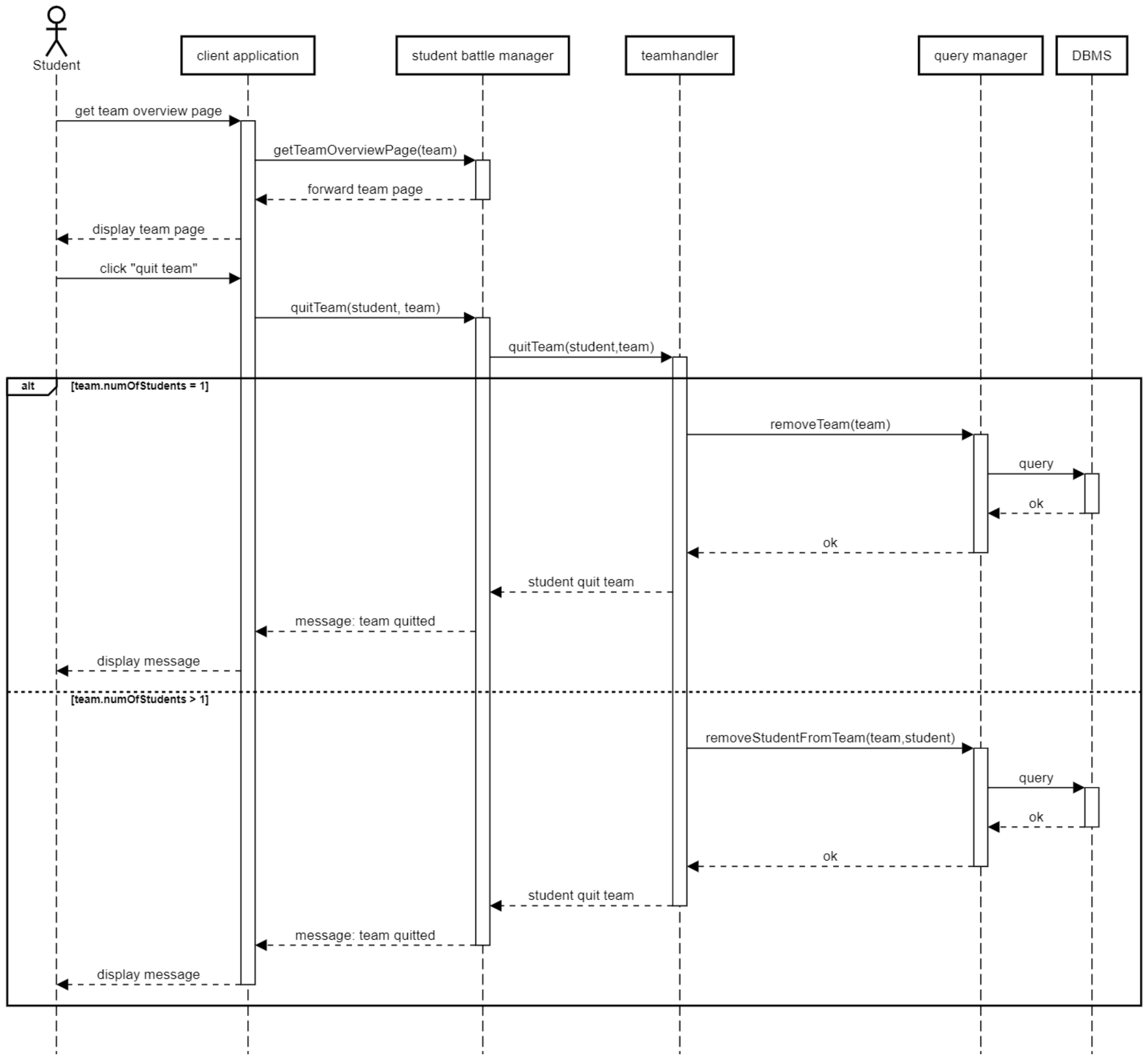


Figure 11: A student quits their team

New code pushed on GitHub repository

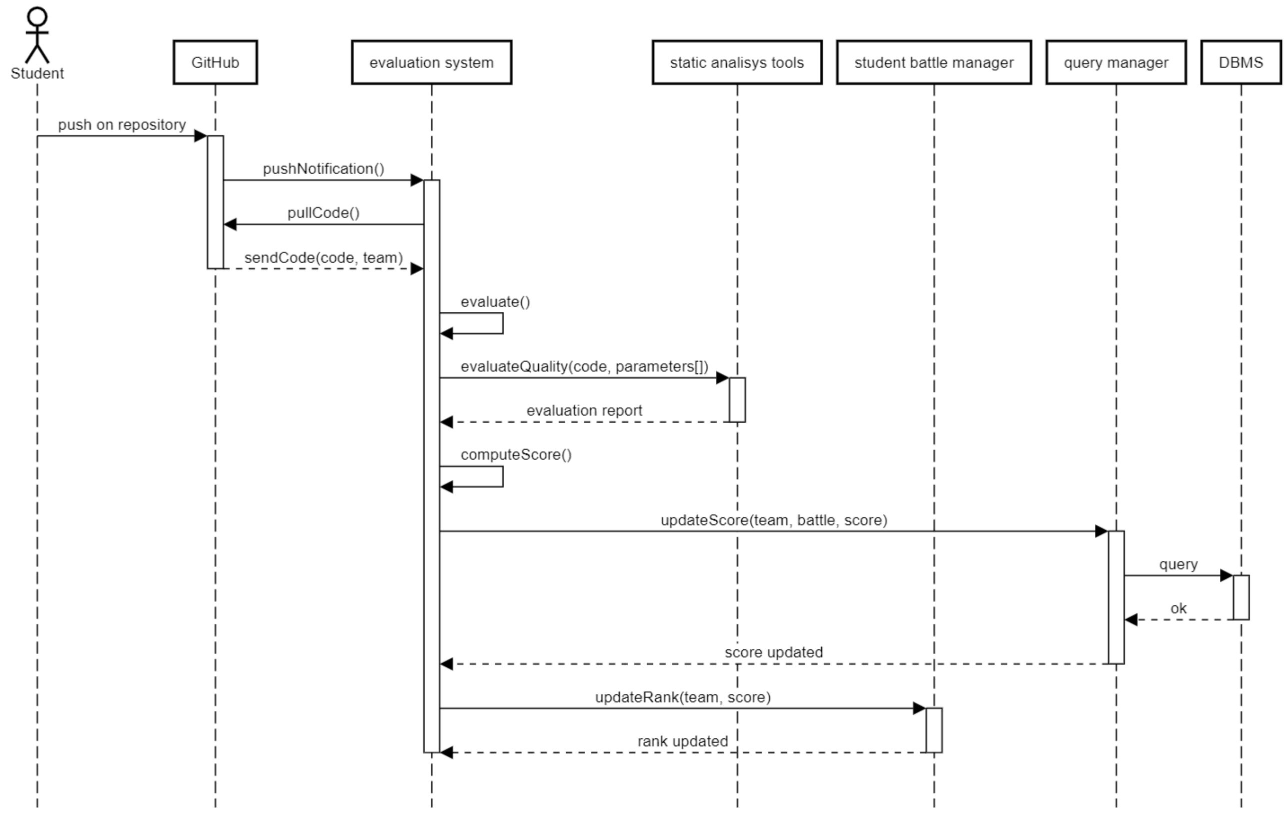


Figure 12: A student executes a push on their GitHub repository for their project

Creation of a Tournament and Badge creation process

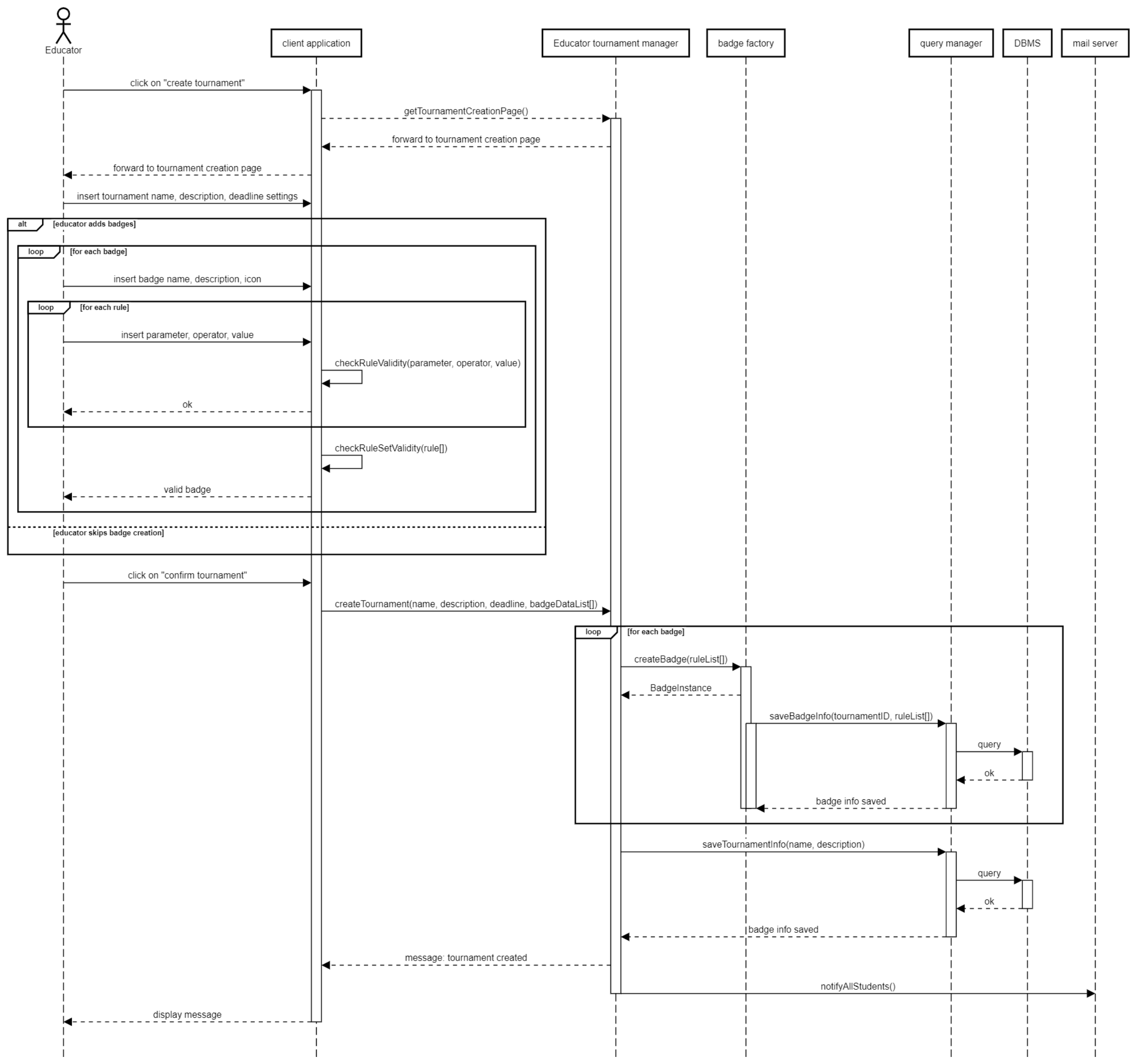
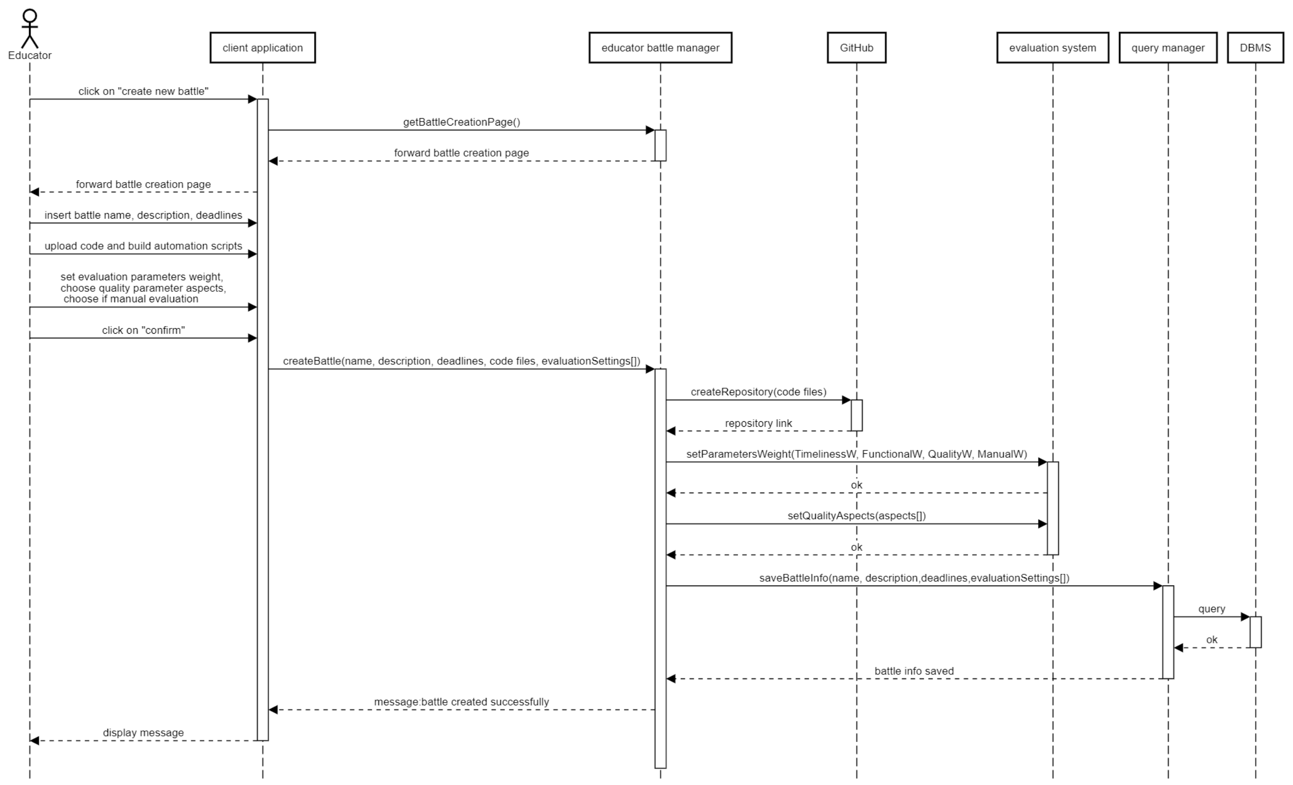

Educator
client application
Educator tournament manager
badge factory
query manager
DBMS
mail server
click on "create tournament"
getTournamentCreationPage()
forward to tournament creation page
forward to tournament creation page
insert tournament name, description, deadline settings
alt [educator adds badges]
loop [for each badge]
insert badge name, description, icon
loop [for each rule]
insert parameter, operator, value
checkRuleValidity(parameter, operator, value)
ok
checkRuleSetValidity(rule[])
valid badge
[educator skips badge creation]
click on "confirm tournament"
createTournament(name, description, deadline, badgeDataList[])
loop [for each badge]
createBadge(ruleList[])
BadgeInstance
saveBadgeInfo(tournamentID, ruleList[])
query
ok
badge info saved
saveTournamentInfo(name, description)
query
ok
badge info saved
message: tournament created
notifyAllStudents()
display message

Figure 13: Educator creates a tournament. Badge creation process.

Battle creation



Educator
client application
educator battle manager
GitHub
evaluation system
query manager
DBMS
click on "create new battle"
getBattleCreationPage()
forward battle creation page
forward battle creation page
insert battle name, description, deadlines
upload code and build automation scripts
set evaluation parameters weight, choose quality parameter aspects, choose if manual evaluation
click on "confirm"
createBattle(name, description, deadlines, code files, evaluationSettings[])
createRepository(code files)
repository link
setParametersWeight(TimelinessW, FunctionalW, QualityW, ManualW)
ok
setQualityAspects(aspects[])
ok
saveBattleInfo(name, description,deadlines,evaluationSettings[])
query
ok
battle info saved
message:battle created successfully
display message

Figure 14: Educator creates a battle

Invite educators to a Tournament

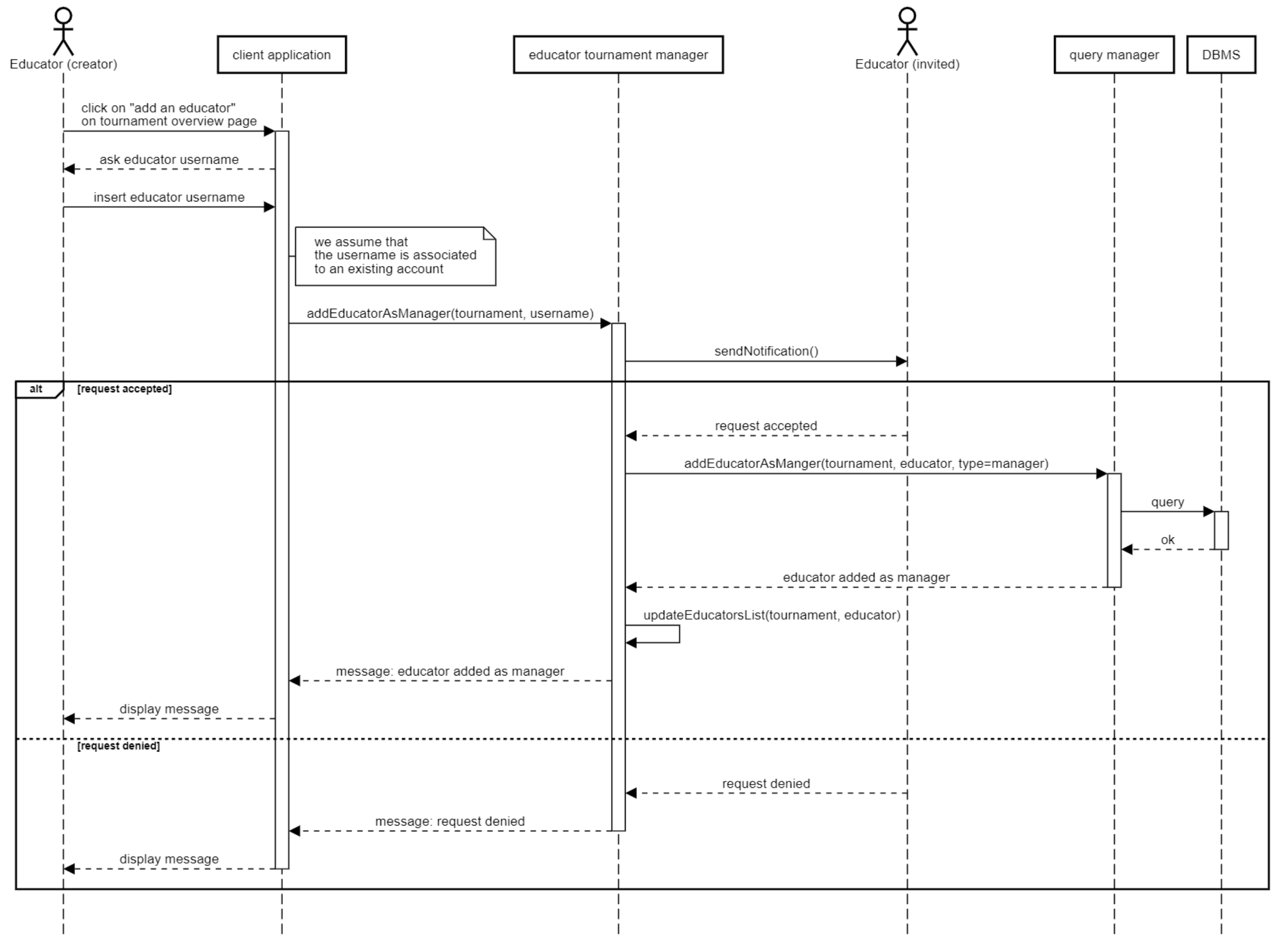

Educator (creator)
client application
educator tournament manager
Educator (invited)
query manager
DBMS
click on "add an educator" on tournament overview page
ask educator username
insert educator username
we assume that the username is associated to an existing account
addEducatorAsManager(tournament, username)
sendNotification()
alt
[request accepted]
request accepted
addEducatorAsManger(tournament, educator, type=manager)
query
ok
educator added as manager
updateEducatorsList(tournament, educator)
message: educator added as manager
display message
[request denied]
request denied
message: request denied
display message

Figure 15: An educator invites another educator as tournament manager

Battle closure

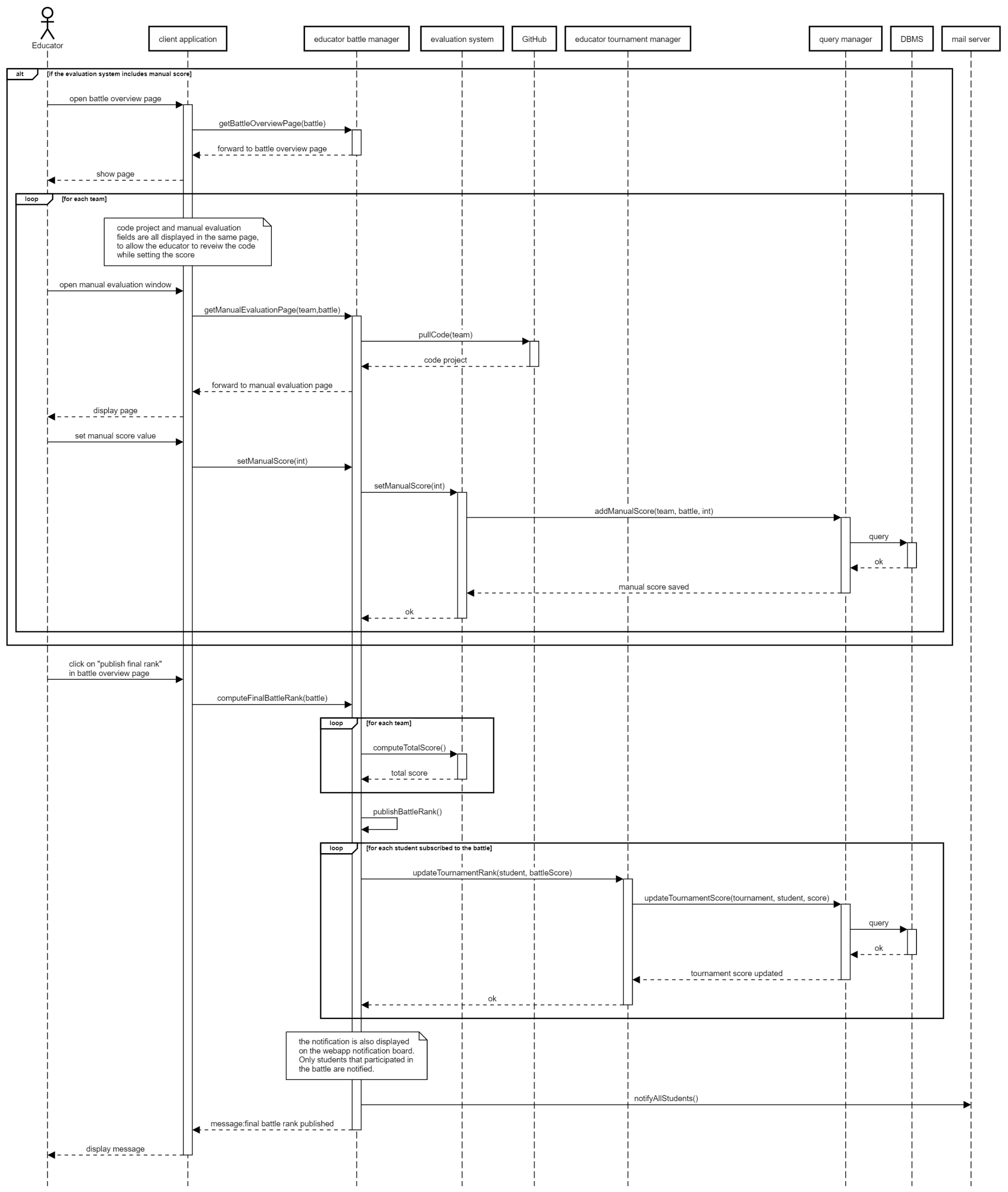

Educator
client application
educator battle manager
evaluation system
GitHub
educator tournament manager
query manager
DBMS
mail server
alt
[if the evaluation system includes manual score]
open battle overview page
getBattleOverviewPage(battle)
forward to battle overview page
show page
loop
[for each team]
code project and manual evaluation fields are all displayed in the same page, to allow the educator to reveiw the code while setting the score
open manual evaluation window
getManualEvaluationPage(team,battle)
pullCode(team)
code project
forward to manual evaluation page
display page
set manual score value
setManualScore(int)
setManualScore(int)
addManualScore(team, battle, int)
query
ok
manual score saved
ok
click on "publish final rank" in battle overview page
computeFinalBattleRank(battle)
loop
[for each team]
computeTotalScore()
total score
publishBattleRank()
loop
[for each student subscribed to the battle]
updateTournamentRank(student, battleScore)
updateTournamentScore(tournament, student, score)
query
ok
tournament score updated
ok
the notification is also displayed on the webapp notification board. Only students that participated in the battle are notified.
notifyAllStudents()
message:final battle rank published
display message

Figure 16: Closing of a battle after deadline expiration

Tournament Closure

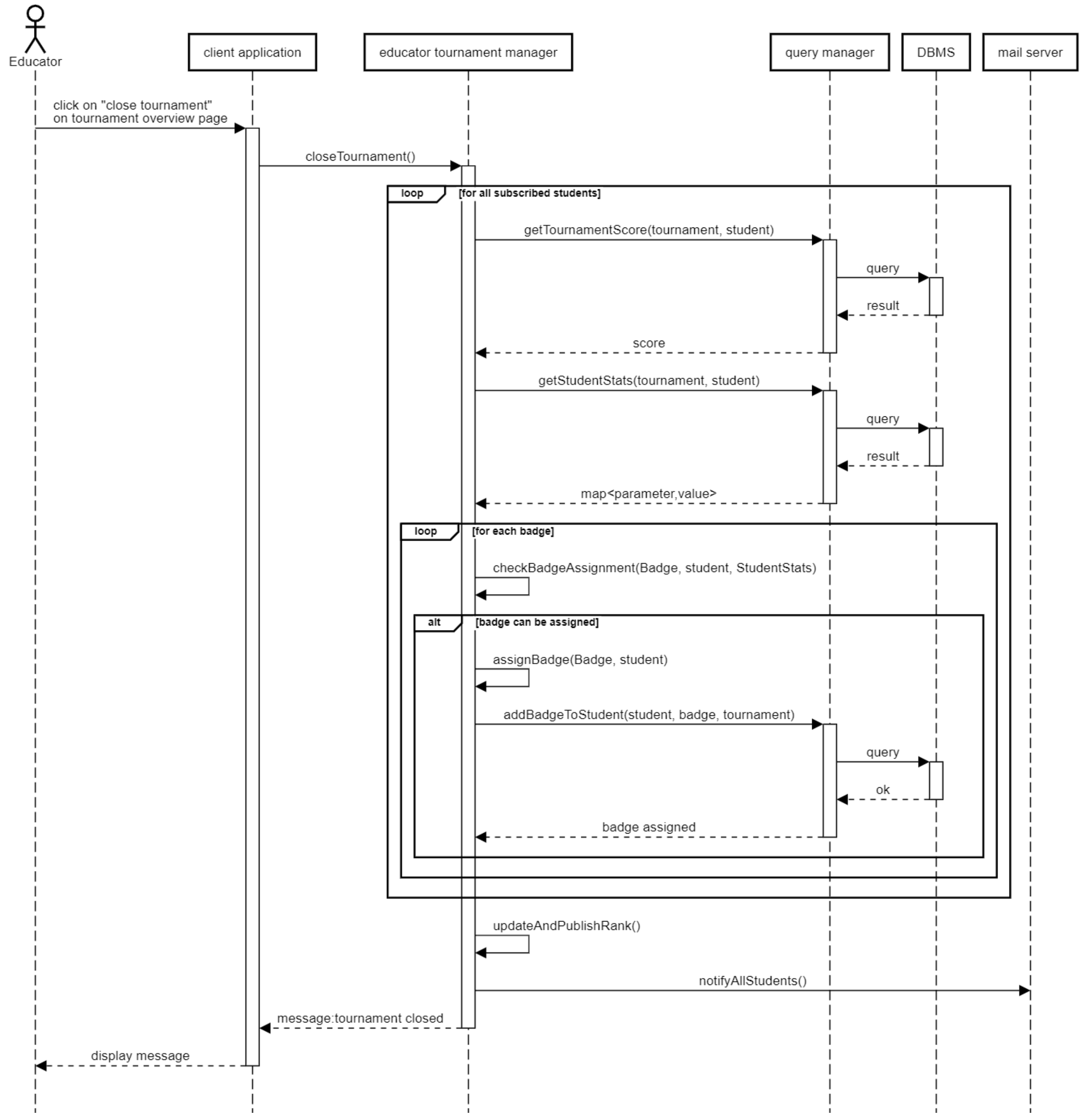

Educator
client application
educator tournament manager
query manager
DBMS
mail server
click on "close tournament" on tournament overview page
closeTournament()
loop [for all subscribed students]
getTournamentScore(tournament, student)
query
result
score
getStudentStats(tournament, student)
query
result
map<parameter,value>
loop [for each badge]
checkBadgeAssignment(Badge, student, StudentStats)
alt [badge can be assigned]
assignBadge(Badge, student)
addBadgeToStudent(student, badge, tournament)
query
ok
badge assigned
updateAndPublishRank()
notifyAllStudents()
message:tournament closed
display message

Figure 17: Closing of a tournament

2.5 Component Interfaces

The following diagram outlines the main methods that can be invoked on the interfaces made available by the various components, referencing the key processes detailed in the runtime view section. It is important to note that the methods presented in the *Component Interfaces diagram* should not be interpreted as the precise methods developers will write. Instead, they serve as a conceptual representation of the capabilities offered by component interfaces.
Note: the arrows' head are pointed towards the component whose interface is being used. So for example, the `Student Tournament Manager` component uses the the `Query Manager` interface.

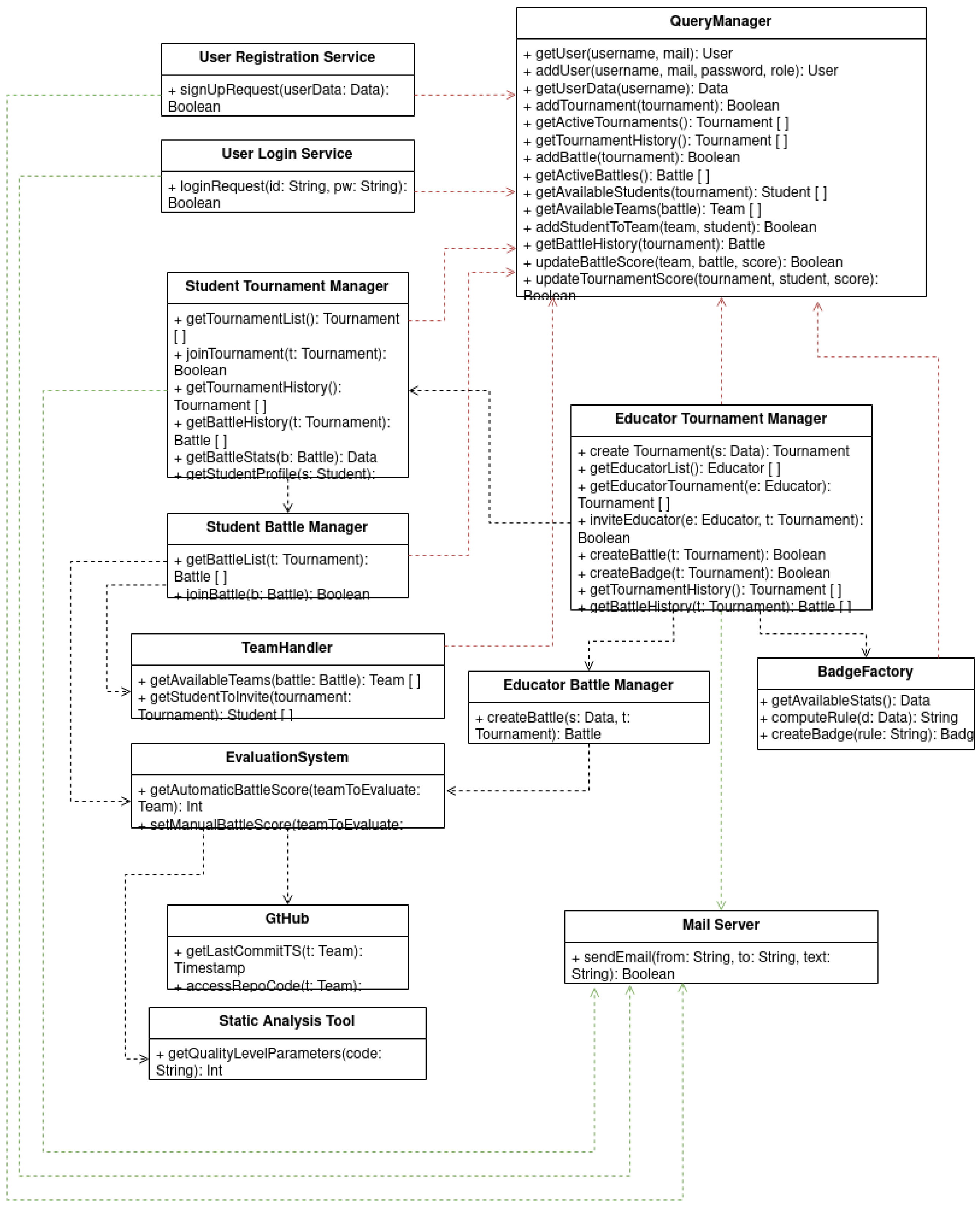

QueryManager
+ getUser(username, mail): User
+ addUser(username, mail, password, role): User
+ getUserData(username): Data
+ addTournament(tournament): Boolean
+ getActiveTournaments(): Tournament [ ]
+ getTournamentHistory(): Tournament [ ]
+ addBattle(tournament): Boolean
+ getActiveBattles(): Battle [ ]
+ getAvailableStudents(tournament): Student [ ]
+ getAvailableTeams(battle): Team [ ]
+ addStudentToTeam(team, student): Boolean
+ getBattleHistory(tournament): Battle
+ updateBattleScore(team, battle, score): Boolean
+ updateTournamentScore(tournament, student, score): Boolean
User Registration Service
+ signUpRequest(userData: Data): Boolean
User Login Service
+ loginRequest(id: String, pw: String): Boolean
Student Tournament Manager
+ getTournamentList(): Tournament [ ]
+ joinTournament(t: Tournament): Boolean
+ getTournamentHistory(): Tournament [ ]
+ getBattleHistory(t: Tournament): Battle [ ]
+ getBattleStats(b: Battle): Data
+ getStudentProfile(s: Student):
Student Battle Manager
+ getBattleList(t: Tournament): Battle [ ]
+ joinBattle(b: Battle): Boolean
Educator Tournament Manager
+ create Tournament(s: Data): Tournament
+ getEducatorList(): Educator [ ]
+ getEducatorTournament(e: Educator): Tournament [ ]
+ inviteEducator(e: Educator, t: Tournament): Boolean
+ createBattle(t: Tournament): Boolean
+ createBadge(t: Tournament): Boolean
+ getTournamentHistory(): Tournament [ ]
+ getBattleHistory(t: Tournament): Battle [ ]
TeamHandler
+ getAvailableTeams(battle: Battle): Team [ ]
+ getStudentToInvite(tournament: Tournament): Student [ ]
Educator Battle Manager
+ createBattle(s: Data, t: Tournament): Battle
BadgeFactory
+ getAvailableStats(): Data
+ computeRule(d: Data): String
+ createBadge(rule: String): Badg
EvaluationSystem
+ getAutomaticBattleScore(teamToEvaluate: Team): Int
+ setManualBattleScore(teamToEvaluate:
GtHub
+ getLastCommitTS(t: Team): Timestamp
+ accessRepoCode(t: Team):
Static Analysis Tool
+ getQualityLevelParameters(code: String): Int
Mail Server
+ sendEmail(from: String, to: String, text: String): Boolean

Figure 18: CKB Platform Component Diagram

2.6 Selected Architectural Styles and Patterns

3-Tier Architecture

As previously discussed, the CKB Platform is structured on a **3-tier Client-Server Architecture**. This architectural choice ensures the system's scalability, modularity, and maintainability, while also guaranteeing platform independence. Additionally, the components within the application server are designed to maintain cohesiveness and exhibit low coupling among modules, with advantages in terms of comprehensibility and modifiability within the system.

Model View Controller (MVC)

The MVC pattern separates the concerns of data management (Model), presentation (View), and request handling (Controller). This separation enhances modularity, maintainability, and flexibility in developing and managing server-side logic. In the context of a 3-tier architecture:

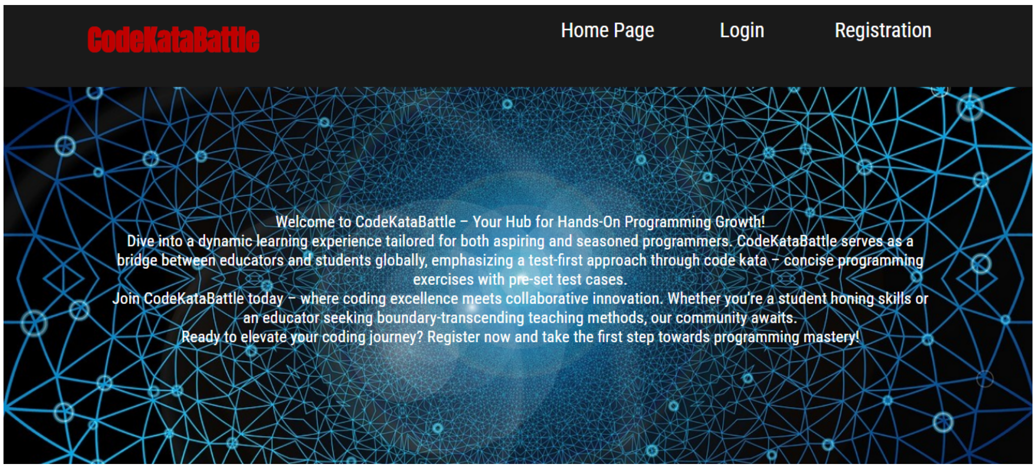
1. The **View** corresponds to the presentation tier, responsible for displaying information to users and receiving their input.
2. The **Controller** corresponds to the business logic tier, managing the flow of data and business rules.
3. The **Model** corresponds to the data storage tier, handling the storage and retrieval of data.

Observer Pattern

This pattern relies on two categories of objects: the **Subjects** and the **Observers**. The Subject is an object that can change its state, while the Observer is subscribed to the Subject. Whenever a Subject change its state the Observer is automatically notified. In the CKB Platform context, this pattern can be used in the `EvaluationSystem` component: whenever a Team commits new code to the Repository associated with a Battle, the `GitHubRepo` object (acting as the Subject) automatically informs the `EvaluationSystem` object (the Observer) about the new commit. This triggers the EvaluationSystem component that will compute the Team's CokeKata evaluation, updating the real-time score for the Team.

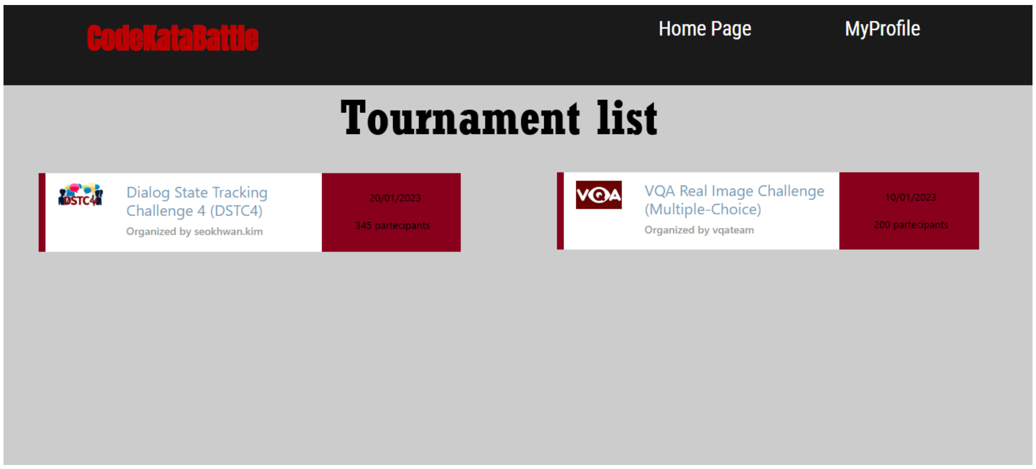
3 User Interface Design

This chapter showcases mockups of the key pages of the CKB WebApp. While predominantly displayed in desktop format, all pages of the WebApp are equally accessible in a mobile format. The design of the Web Application interface adheres to responsive principles, ensuring optimal usability on mobile devices as well.



Welcome to CodeKataBattle – Your Hub for Hands-On Programming Growth!
Dive into a dynamic learning experience tailored for both aspiring and seasoned programmers. CodeKataBattle serves as a bridge between educators and students globally, emphasizing a test-first approach through code kata – concise programming exercises with pre-set test cases.
Join CodeKataBattle today – where coding excellence meets collaborative innovation. Whether you're a student honing skills or an educator seeking boundary-transcending teaching methods, our community awaits.
Ready to elevate your coding journey? Register now and take the first step towards programming mastery!

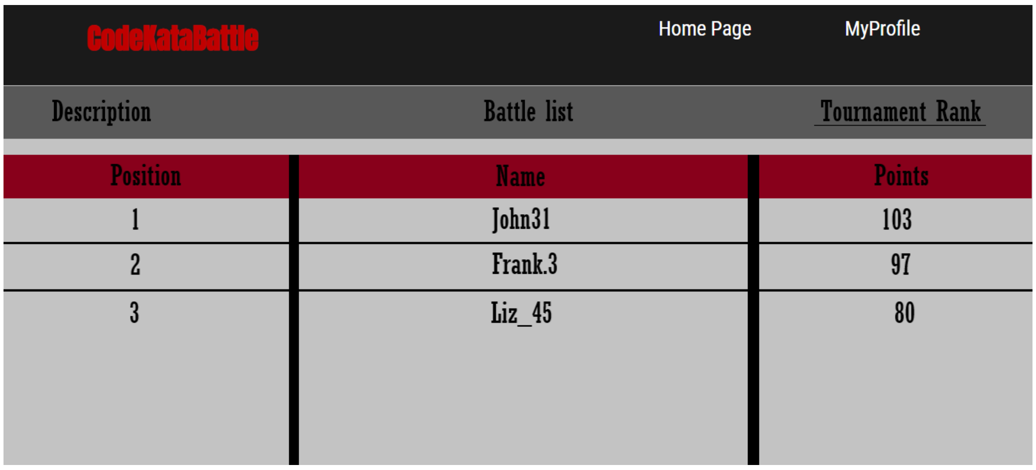
Figure 19: CKB Home Page



Tournament list

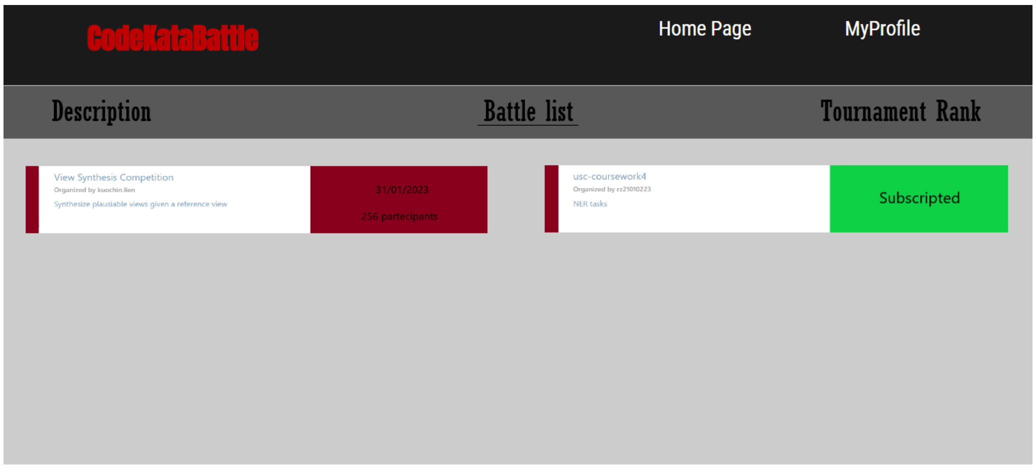
Dialog State Tracking Challenge 4 (DSTC4)	Organized by seokhwan.kim	20/01/2023 345 partecipants
VQA Real Image Challenge (Multiple-Choice)	Organized by vqateam	10/01/2023 200 partecipants

Figure 20: This is the page where the student can join one or multiple available tournaments



Position	Name	Points
1	John31	103
2	Frank.3	97
3	Liz_45	80

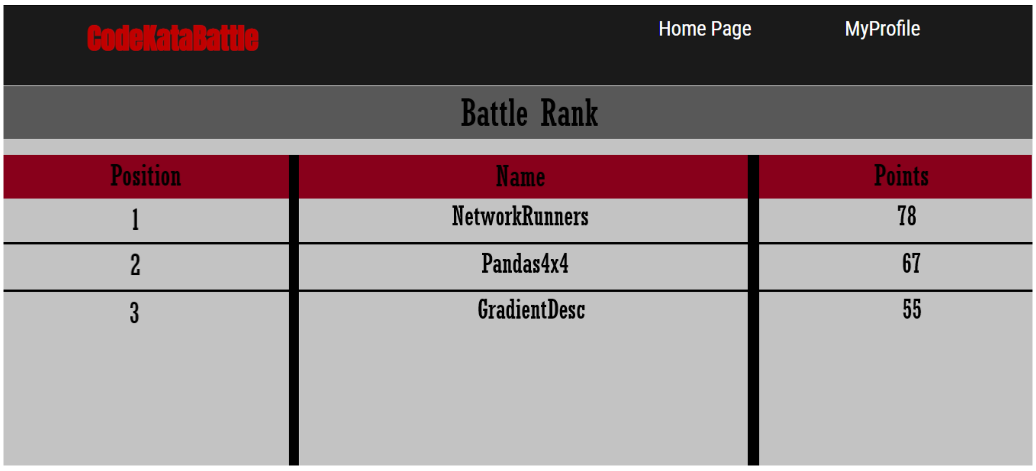
Figure 21: CKB Tournament Rank



View Synthesis Competition
Organized by kuochin.lien
Synthesize plausiable views given a reference view
31/01/2023
256 partecipants

usc-coursework4
Organized by rz21010223
NER tasks
Subscripted

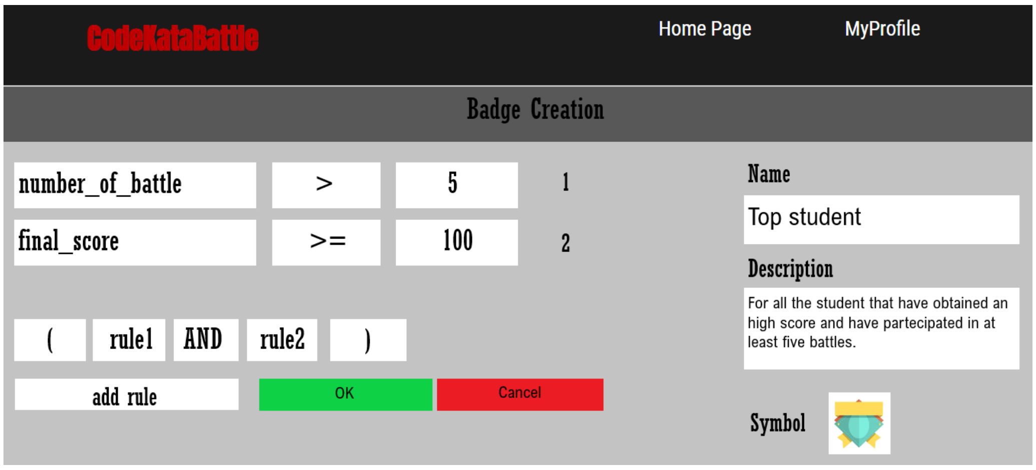
Figure 22: This is the page where the student can join one or multiple battles in a tournament he has joined



Battle Rank

Position	Name	Points
1	NetworkRunners	78
2	Pandas4x4	67
3	GradientDesc	55

Figure 23: CKB Battle Rank



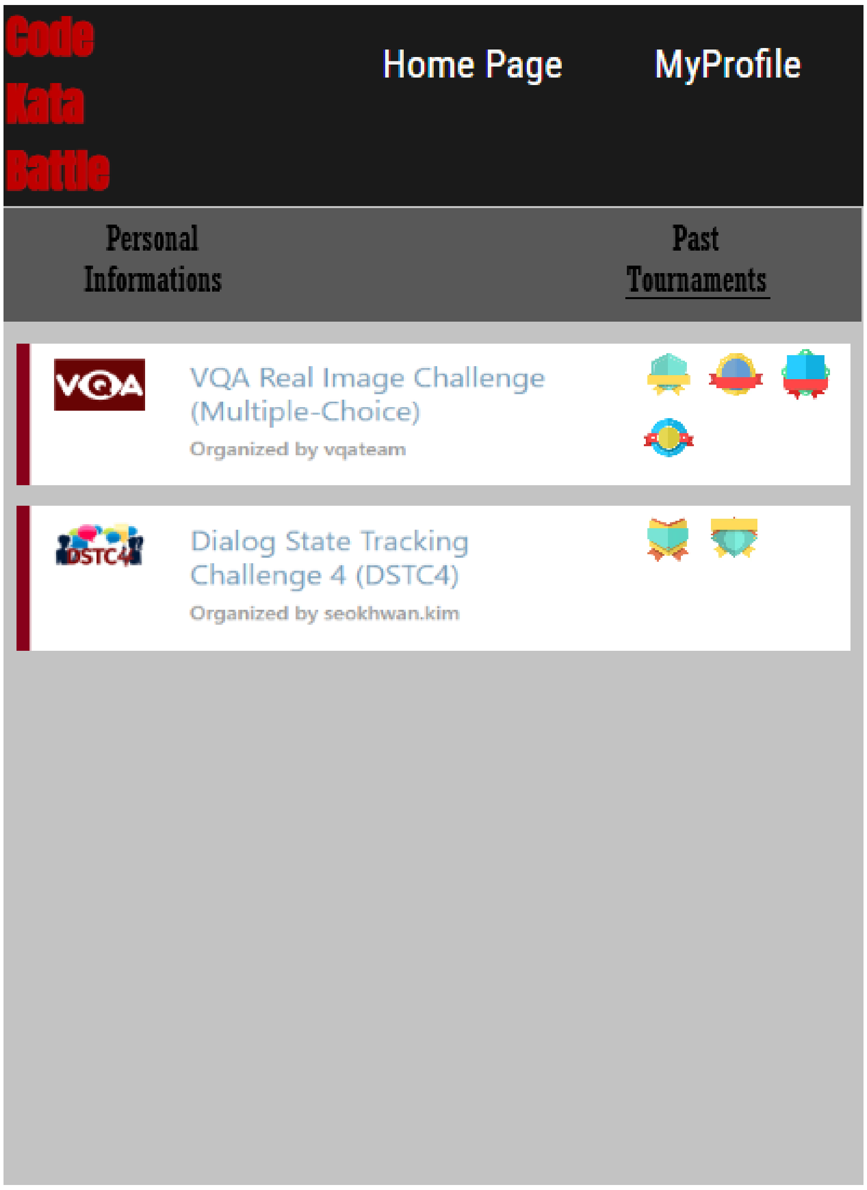
number_of_battle	>	5	1
final_score	>=	100	2

(rule1 AND rule2)

Name: Top student

Description: For all the student that have obtained an high score and have partecipated in at least five battles.

Figure 24: This is the page where the educator can set the rules for a badge during the tournament creation process



VQA Real Image Challenge (Multiple-Choice)

Organized by vqateam

Dialog State Tracking Challenge 4 (DSTC4)

Organized by seokhwan.kim

Figure 25: This is the page where are presented all the tournaments that a student has partecipated and all the badges he has won

4 Requirement Traceability

Components	Requrements
WebApp	R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, R16, R17, R18, R19, R20, R21, R22, R23, R24, R25, R26, R27, R28, R29
User registration service	R1
User login service	R2
Educator Tournament Manager	R4, R5, R6, R7, R8, R9, R11, R27, R28, R29
Educator battle manager	R11, R12, R13, R14, R15, R16, R25, R26
BadgeFactory	R6
Student tournament manager	R10, R24
Student Battle Manager	R17, R18, R19, R20, R21, R22, R23, R24
Evaluation system	R24
TeamHandler	R17, R18, R19, R20, R21, R22, R23
Static Analysis Tool	R24, R25
Github	R24
Mail server	R9, R16, R18, R20, R26, R28
Query manager and Database	R1, R2, R3, R4, R5, R6, R7, R8, R10, R11, R12, R13, R14, R15, R17, R19, R21, R22, R23, R24, R25, R26, R27, R28, R29

The following list shows the **Functional Requirements** of the CKB system, which are the same reported int he RASD document:

R1 : The system must allow an unregistered user to register an account with personal informations and define a username.

R2 : The system must allow a registered user to login into his account.

R3 : The system must allow registered educators to create a new tournament.

R4 : The system must allow registered educators to write a description for the tournament during a tournamnet creation process.

R5 : The system must allow registered educators to set a subscription deadline during a tournament creation process.

R6 : The system must allow registered educators to define badge's names and rules during a tournamnet creation process.

R7 : The system must allow registered educators to give permission to other educators to create battles in the tournament they have created.

R8 : The system must allow registered educators to accept or deny the permission for the creation of battles by another educator.

R9 : The system must allow registered students to be notified (also via e-mail) when a new tournament is created.

R10 : The system must allow registered students to subscribe to a new tournament until the registration deadline expires.

R11 : The system must allow registered educators with permission to create a new battle within an existing tournament.

R12 : The system must allow registered educators to upload the code kata during a battle creation process.

R13 : The system must allow registered educators to set minimum and maximum team's members number during a battle creation process.

R14 : The system must allow registered educators to set a submission deadline and a finish deadline during a battle creation process.

R15 : The system must allow registered educators to set scoring parameters during a battle creation process.

R16 : The system must allow registered students to be notified (also via e-mail) when a new battle is created in a tournament they are involved.

R17 : The system must allow registered students to form teams for a battle.

R18 : The system must allow registered students to invite other students to join their team.

R19 : The system must allow registered students to to accept or deny the invite by another student to be part of his team.

R20 : The system must allow registered students to send a request to join an existing team for a battle they want to join.

R21 : The system must allow registered students to accept or deny a request to join their team by another student.

R22 : The system must allow registered students to quit their team before joining the battle.

R23 : The system must allow registered students to join battles within tournaments they're subscribed to.

R24 : The system must allow registered students to track their score every time they perform a push on the GitHub repository of their project.

R25 : The system must allow registered educators to perform the manual evaluation of the codes and assign a personal score during the consolidation stage.

R26 : The system must allow registered students to be notified (also via e-mail) when the final battle rank is available.

R27 : The system must allow registered educators to close tournaments (if they are the one that have created the tournament).

R28 : The system must allow registered students to be notified (also via e-mail) when the final tournament rank is available.

R29 : The system must allow registered users to visualize badges obtained by the students (in their personal profile and also in the final tournament rank).

5 Implementation, Integration and Test Plan

5.1 Introduction

Describing how and the order of the implementation is crucial to be able to develop a well structured software. Unit testing and integration testing are also fundamental to avoid bugs, errors and to check if all the functionalities are well implemented.

5.2 Implementation Plan

A bottom-up strategy is the best way to develop this kind of software, starting from the low level components, like the QueryManager, and ending with the WebApp. The following list shows the order in which the internal components must be implemented:

- Data
- QueryManager
- Tournament Management Service:
  - Evaluation System
  - TeamHandler
  - Student Battle Manager
  - Student Tournament Manager
  - BadgeFactory
  - Educator Battle Manager
  - Educator Tournament Manager
- Access Manager Service
  - User Login Service
  - User Registration Service
- WebApp

The external components are not presented since they are assumed to be already implemented.

5.3 Integration Strategy

The following diagrams show step by step how each component of the system is implemented and integrated with the others, according to the order described above. For clarification, the arrows in the diagrams below indicate the relationships between the varius components and the implementation status of the relationship. A dashed arrow indicates that the relationship is being implemented at that stage of implementation. Instead, the solid arrow indicates that the relationship has already been implemented. In each integration step it is necessary to run integration tests to check if the integration issuccessful.

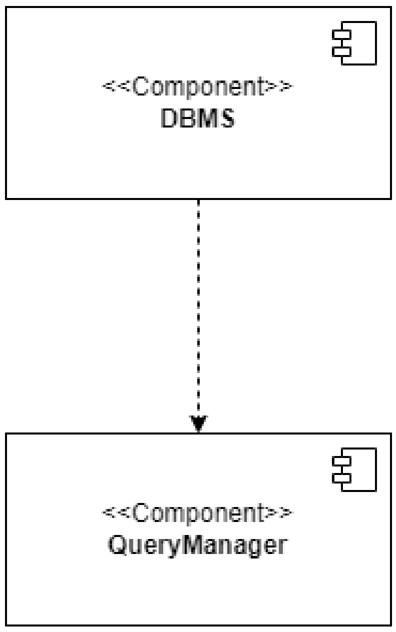


Figure 26: The implementation and testing of the QueryManager component

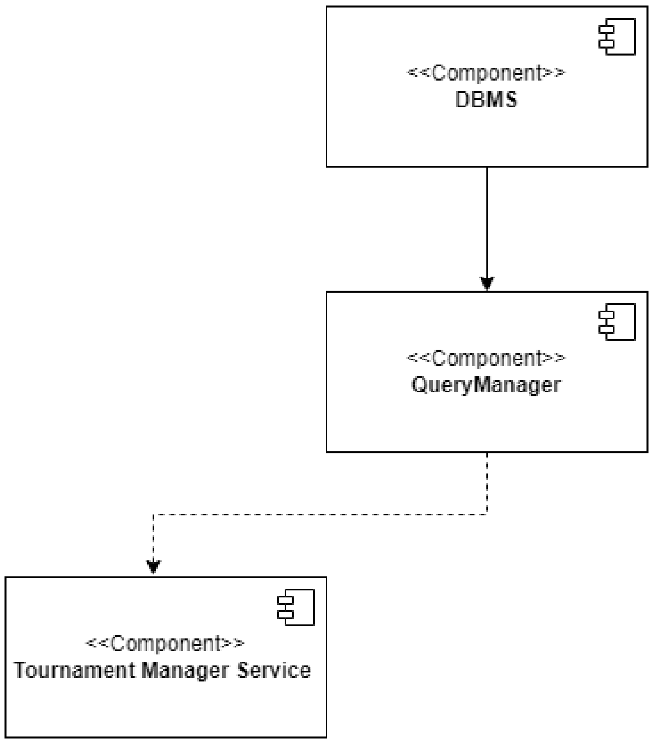


Figure 27: The implementation and testing of the Tournament Manager Service component

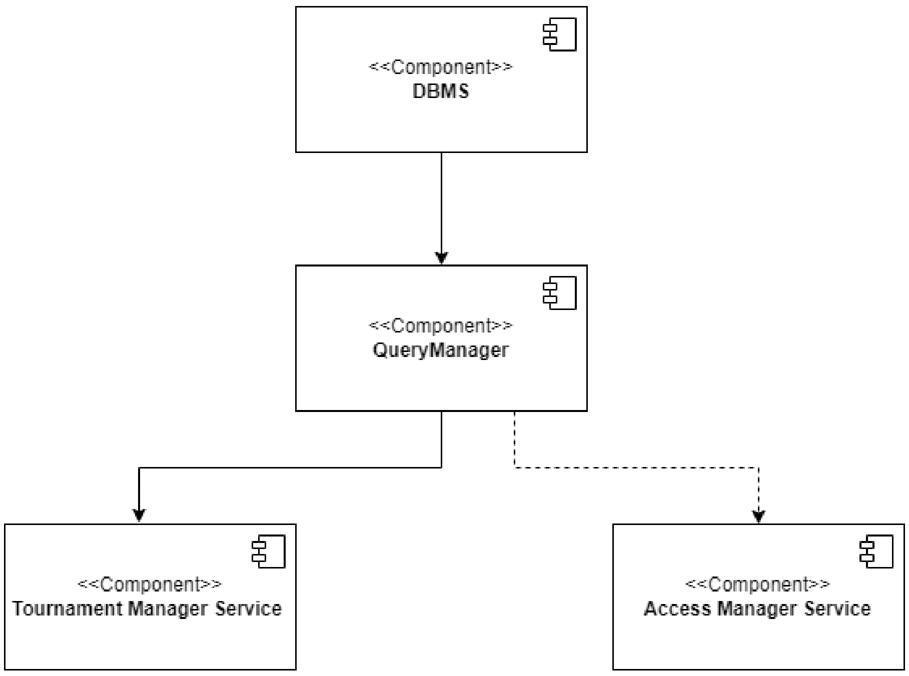


Figure 28: The implementation and testing of the Login Manager Service component

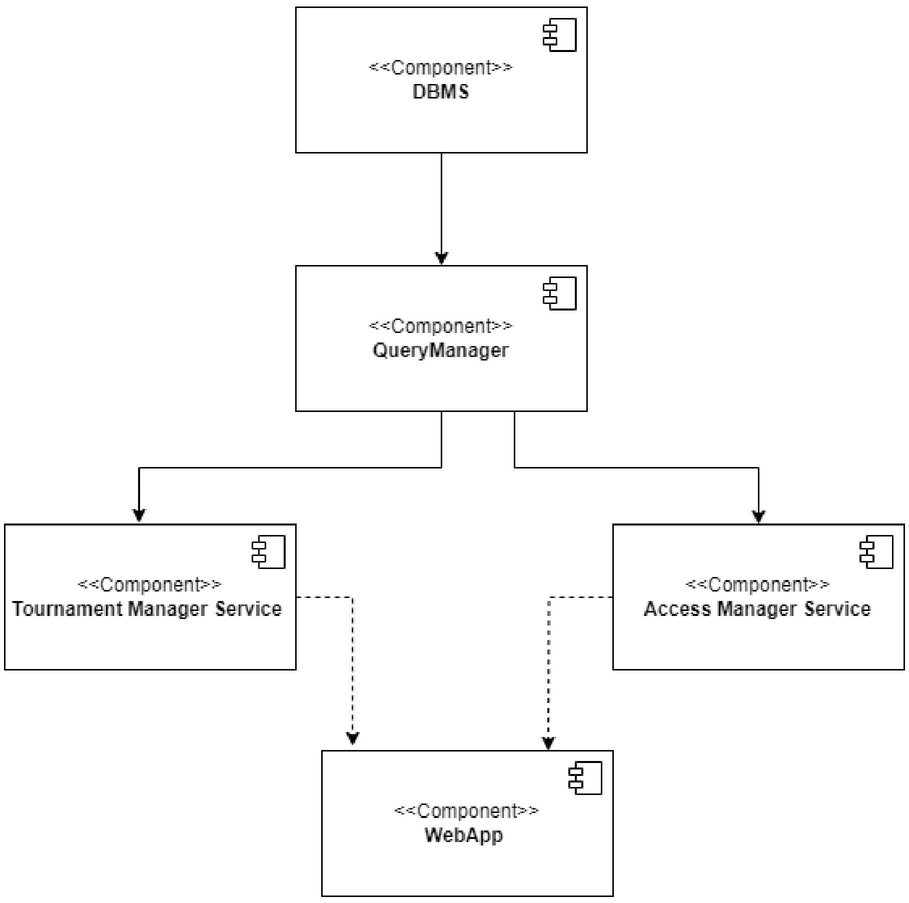


Figure 29: The implementation and testing of the WebApp component

5.4 System Testing

When the system is fully developed and all the integration tests are successfully passed, it is necessary to verify the system correctness with the system testing. The system testing concerns the functional and non-functional requirements described in the RASD and the performance evalutation. The system testing is divided in:

- Functional testing: To evaluate the compliance of the system with specified functional and non-functional requirements.
- Performance testing: To determine how the system performs in terms of scalability, reliability and resource usage. It also identify bottlenecks affecting response time, utilization, thoughput.
- Load testing: To measure the response of putting demand on the system. It shows bugs such as memory leaks, mismanagement of memory, buffer overflows.
- Stress testing: To measure the robustness, availability and error handling by testing beyond the limits of normal operation.

6 Effort Spent

The following tables display the effort spent by each member of the group.

Daniele Gagni

Chapter	**Hours**
Introduction	1h (1h together)
Architectural Design	12h (4h together)
User Interface Design	3h (3h together)
Requirements Traceability	2h (2h together)
Implementation, Integration and Test Pkan	0.5h (0.5h together)

Guglielmelli Isabella

Chapter	**Hours**
Introduction	1.5h (1h together)
Architectural Design	13h (4h together)
User Interface Design	3h (3h together)
Requirements Traceability	3h (2h together)
Implementation, Integration and Test Pkan	0.5h (0.5h together)

Mariotti Marco

Chapter	**Hours**
Introduction	1h (1h together)
Architectural Design	4h (4h together)
User Interface Design	13h (3h together)
Requirements Traceability	2h (2h together)
Implementation, Integration and Test Pkan	1.5h (0.5h together)

7 References

- SonarQube
- GitHub