Immagine che contiene testo, Carattere, logo, Elementi grafici

Descrizione generata automaticamente

SOFTWARE ENGINEERING II

CKB – CodeKataBattle

Design Document

Version 1.0

Polito Attilio

Rigione Pisone Raimondo

Soricelli Francesco

*January 7, 2024*

**Nessuna voce di sommario trovata.**

# 1. Introduction

## Purpose

The aim of the CodeKataBattle (CKB) platform is to provide a dynamic and engaging environment for students to improve their software development skills through collaborative learning. The project aims to foster a culture of continuous improvement using coding challenges and competitions in a structured educational environment. The main goals of the project are:

1. Skill Development: CodeKataBattle promotes a test-first approach to make students improve their programming skills by challenging them with diverse programming exercises.
2. Collaborative learning: Encourage teamwork and collaboration between students by forming teams and allowing them to participate in coding battles. This collaborative approach not only strengthens technical capabilities, but also promotes effective communication and problem-solving skills.
3. Define Educators’ role: Provide educators with a platform to create, manage, and evaluate coding battles. Educators can create tournaments, give permissions to other educators to manage them, design badges for the gamification aspect, and guide students to hone their coding skills through their Manual Evaluations relative to students performances.
4. Automated Evaluation: Implement a robust automated evaluation system to evaluate code quality, functional aspects, and timeliness. Integration with static analysis tools ensures comprehensive evaluation and timely feedback, motivating students to iterate and refine their solutions.
5. Competition and Ranking: Introduce competitive elements to stimulate students' motivation. By assigning scores during battles and creating real-time ranks, the platform promotes a sense of accomplishment and healthy competition among participants.
6. Gamification: Use gamification elements, such as badges, to enhance the learning experience. These badges provide additional motivation for students to excel in various aspects of coding challenges, to recognize and celebrate individual achievements.
7. Tournament Structure: Organize coding battles into tournaments, allowing students to accumulate scores in multiple challenges and visualize tournament informations.
8. Notification system: Implement a notification system that facilitates effective communication with the platform. Students and Educators will receive timely updates on upcoming battles, deadlines and tournament results to provide information and maintain engagement.

Through these goals, CodeKataBattle platform not only improves technical skills, but also provides continuous learning and improvement for the students in the field of software development.

## 1.2 Scope

The CodeKataBattle (CKB) project demands innovative solutions to effectively train students, preparing them for the challenges of the professional world, with the increasing importance of skills in software development and the increasing complexity of the industry providing students with an interactive and collaborative environment to improve their abilities. CKB provides a platform that allows students to refine their skills through practical and exciting programming challenges.

The platform has 2 main actors - students and educators - who are actively involved, each playing a unique role. Students who make up the active components of the system participate in the Battle by registering individually or forming teams based on specific participation requirements. When a student joins the battle, after the battle starts, he will receive a notification with a link to the assigned code kata GitHub repository for his Team. Here, follow the test-driven development (TDD) approach, he’ll complete the exercise and submit the solution by the specified deadline.

Educators, on the other hand, act as architects of competitions. They have a role in creating tournaments and defining combat specifications, including details such as code type descriptions, programming languages, tests to pass, registration and submission deadlines. An Educator can also set mandatory evaluation parameters, such as test code quality, and optional evaluation parameters, such as the manual evaluation feature. After an Educator has created a battle, the platform manages the student enrollment process, distributes GitHub repositories, and automates evaluation through GitHub Actions.

During the competition, the system dynamically assigns scores to the team based on automatic evaluation (test success, timeliness of submission, quality of code, etc.) and manual evaluation by educators. The scores accumulated in each battle contribute to each student's overall tournament score. To further enrich the experience, the platform has introduced a gamification system through the awarding of badges. Educators define rules and variables associated with badges that are assigned to students based on their performance during the tournament. These gamification badges will be displayed in the student's profile.

Joining CodeKataBattle not only improves technical skills, but also promotes collaboration, problem solving, and compliance with best practices in software development. This platform prepares students for the challenges they face in their professional careers. In addition, the gamification element adds an exciting layer to the learning process and motivates students to strive for excellence.

Going forward, the CKB team expects continuous improvement and expansion of the platform. Future plans include considering additional features, improving the evaluation process, and incorporating feedback from both students and educators to ensure an optimal learning experience. The goal is a major platform to hone the skills of the next generation of software developers evolving with the ever-changing landscape of software development education.

## 1.3 Definitions, Acronyms, Abbreviations

### 1.3.1 Definitions

|  |  |
| --- | --- |
| **Definition** | **Description** |
| CodeKataBattle (CKB) | The online platform designed to hold coding competitions and challenges to improve students’ software development abilities. |
| Code Kata | The exercise that includes a brief textual description and a software project with build automation scripts that contains a set of test cases. |
| Test-first or test-driven development (TDD) approach | A software development model that necessitates writing automatic tests before writing the software that needs to be tested. The development of the application software is solely focused on passing the previously written automatic tests. |
| Tournament | Organization on CKB created by an educator, consisting of a series of challenges. |
| Battle | Challenge or competition that is developed by an educator and held during a tournament. |
| Team | A group of students coming together to take part in a particular CKB battle. Students create teams on the platform before starting a battle. |
| GitHub repository | A distributed version control system and digital storage area powered by Git. In this instance, the “code kata” refers to the CKB platform programming project linked to a particular battle, and  it can be found in the GitHub repository. For the battle, every team has a dedicated GitHub repository where students can work together to track changes, solve problems, and version code. |
| GitHub Action | GitHub provides an automation service that lets developers automate software workflows. In this case, GitHub Actions are useful to set up an automated workflow between GitHub and the Platform. This workflow is used in the competition to send an API notification to the CKB platform after every commit, enabling the team’s score to be automatically updated and computed based on the most recent changes made to the code. |
| Score | A natural numerical value between 0 and 100 that is established by accounting for some required factors that are assessed entirely automatically and other optional factors that are assessed manually by educators. |
| Manual evaluation | A subjective evaluation component made by educators who look over and assess students’ work during the consolidation phase. This stage provides a chance to assess factors that might not be picked up by an automated evaluation and that might be more specialized or unique. |
| Notification system | The automated system that alerts teachers and students to impending fights, deadlines, and tournament outcomes. |
| Gamification badges | Virtual awards that educators define as a representation of each student’s achievements during a competition. |
| Valid credentials | The term valid credentials is referred to the email and the password inserted by a user during the login process; they are said to be valid if they correspond to the one inserted by such user during the successful registration process. |
| Conflicting data | The email address inserted during the registration process by a user is defined as a conflicting data if there are already others users of the CKB application that have registered with the same address |
| Tournament Management Permissions | An educator can manage a tournament (creating battles, closing the tournament…) if he has created the tournament or another educator has given him the permissions though the “granting permission” function. |

Table 5: Table of Definitions

### 1.3.2 Acronyms

|  |  |
| --- | --- |
| **Acronym** | **Description** |
| DD | Design Document |
| RASD | Requirement Analysis and Specificatio Document |
| API | Application Programming Interface |
| CKB | CodeKataBattle |
| TDD | Test-Driven Development |
| DBMS | Database Management System |
| HTTP | Hypertext Transfer Protocol |
| TCP-IP | Transmission control protocol – Internet protocol |

Table 6: Table of Acronyms

### 1.3.3 Abbreviations

|  |  |
| --- | --- |
| **Abbreviation** | **Description** |
| Rn | n-th functional requirement |

Table 7: Table of Abbreviation

## 1.4 Revision history

Version 1.0 - 07/01/2024

## 1.5 Reference documents

1. Specification document: R & DD Assignment A.Y. 2022-2023;
2. Course slides;
3. RASDs of the previous academic years;
4. Given links.

## 1.6 Document structure

**Section 1:** In this section it is presented an introduction of the project. Initially, it is discussed the purpose of the system and the goals that it aims to reach. Then, it is described its scope. Moreover, the problem it has been examined by means of a list of world, shared and machine phenomena that represent the system. Finally, it is specified a list of definitions, acronyms and abbreviations that are used in the document.

**Section 2:** Here it is presented an overall description of the project, analyzing multiple scenarios and presenting the UML class diagram that describes the different entities of the system. Furthermore, the use of some statecharts helps to understand what are the actions that can be made through the system and how they happen in a sequence. Finally, it is described more in detail which are the functionalities of the system and the characteristics of the users that interact with it.

**Section 3:** This section contains the specification of the interface requirements, i.e. the definition of the user, hardware, software and communication interfaces needed for the system. Moreover, the functional requirements necessary for the system to work as desired and to be consistent are defined. It is also specified which of these requirements are needed for each goal to be reached. Then, the requirements are analyzed through the use of use cases and their respective diagrams that represent all the possible scenarios described in the previous section.

**Section 4:** This section includes the alloy model and the discussion of its purpose. The results obtained by running the model are properly justified, presenting screens of the generated worlds with the respective comments.

**Section 5:** This section indicates the effort spent by each group member while working on the project.

**Section 6:** this section includes the references.

# Architectural Design

## Overview

The CodeKataBattle (CKB) web application is designed with three key components to deliver its functionalities effectively. These components include a client, a server, and a database. The client, tailored for compatibility with web browsers on PCs, serves the needs of both students and educators engaging in coding challenges. The server acts as the central hub for communication among clients, manages interactions with the internal database, and interfaces with external data sources through APIs. To ensure load distribution and scalability, a load balancer is implemented. The internal database stores diverse information contributed by users and periodically integrates data obtained from external sources.

The CKB web application adopts a three-tier architecture, dividing the system into three distinct groups of components, each serving a specific purpose:

1. **Client Tier:**
   * This tier houses components responsible for views and controllers, defining the user interfaces for students and educators accessing the web application via their PCs.
   * User interfaces obtain necessary data consistency through communication with the application logic tier, minimizing data allocation on the client side.
   * Following a thin client model, this component remains connected to the server to deliver the system's functionalities.
2. **Application Logic Tier:**
   * This tier encapsulates all application logic necessary for processing data collected from the client tier and retrieved from the data tier.
   * It can perform operations such as adding, deleting, or modifying data in the data tier, utilizing API calls for communication.
   * Interacts with external services through API calls to acquire relevant information for the system, subsequently copying it into the internal database.
3. **Data Tier:**
   * Corresponding to a database, this tier houses all data relevant to the CKB web application's scope.
   * Data includes information from external services , user-provided information, and user credentials.
   * The application tier accesses the data tier whenever information retrieval or modification is required.

This three-tier architecture ensures a structured and modular design for the CodeKataBattle web application, promoting scalability, maintainability, and effective separation of concerns across different layers of the system, specifically tailored for PC users.

## 2.2 Component View

### 2.2.1 Class Diagram

In this section, it is reported the class diagram, which has been already analyzed in the RASD document. In this version of the diagram, there have been added the data types for each entity and some attributes that were considered to be important for the implementation of the application, such as the “role” attribute for the user.

Immagine che contiene schermata, diagramma, design

Descrizione generata automaticamente

The "role" attribute has been introduced into the diagram for a User; it is set to true if the user is an educator and false if the user is a student.

This update has been made also in the RASD document by creating a second version of it.

The "staticAnalysisConfig" attribute of the SoftwareProject corresponds to a string containing all parameters for its automatic evaluation.

For simplicity, rules and their associated variables are considered directly within the badges, with variables treated as strings and rules as a list of variables.

Finally, the methods that will implement the several functionalities of the application are not represented; however, they will be analyzed in a diagram that will combine the class diagram and the component diagram, which represents all the possible services.

### 2.2.2 Component Diagram

Immagine che contiene testo, diagramma, Parallelo, Piano

Descrizione generata automaticamente

Immagine che contiene testo, diagramma, ricevuta, Parallelo

Descrizione generata automaticamente

Immagine che contiene testo, diagramma, ricevuta, Piano

Descrizione generata automaticamente

**CopyManager Interface:**

The CopyManager Interface orchestrates the copying of data into the internal database. The primary method, **copyInfo()**, universally returns a generic **Object**. To diversify implementations, the interface introduces the **AutomaticEvaluationCopyManager** that specializes in copying information pertaining to automatic evaluations. The **copyInfo()** method in this implementation specifically returns an instance of **AutomaticEvaluation**.

**InformationService Interface:**

The InformationService Interface is tailored to retrieving and presenting diverse information types for user visualization. Methods such as **getStudentTournamentLeaderboard** and **getStudentBattleLeaderboard** offer insights into tournament and battle leaderboards for individual students. Additionally, **getTeamBattleLeaderboard** provides team-based leaderboard information. The method **getTournamentInformations** delivers a list of battles associated with a particular tournament.

**UserLoginService Interface:**

UserLoginService Interface serves the purpose of facilitating user authentication and registration processes. The **login** method assesses the validity of user credentials, while **checkRegCredentials** ensures the integrity of parameters during registration. The interface further supports processes like sending a confirmation email (**sendConfirmationEmail**) and user registration (**registration**), catering to a seamless user experience.

**NotificationService Interface:**

Dedicated to user communication, NotificationService enables users to send messages and retrieve notifications. The **sendMessage** method allows users to send messages to a specified recipient, and **getNotifications** retrieves a list of notifications for a given user.

**ScoreService Interface:**

ScoreService Interface handles the evaluation of student and team scores. The methods **evaluateStudentScore** and **evaluateTeamScore** are central to assessing individual and team performances, respectively. These evaluations involve scores, automatic evaluations, and manual evaluations.

**TournamentService Interface:**

TournamentService Interface focuses on managing tournaments. Methods like **createTournament** initiate the creation of tournaments, specifying details such as name, description, registration deadline, and ending date. The **joinTournament** method allows students to participate, and **getActiveTournaments** and **getCompletedTournaments** retrieve lists of active and completed tournaments.

**BattleService Interface:**

BattleService Interface is designed for handling battles within the system. The **createBattle** method initiates the creation of battles, specifying details like name, maximum members for a team, registration deadline, and submission deadline. Other methods, such as **joinBattle** and **inviteTeammate**, facilitate student participation and teammate invitations.

**BadgeService Interface:**

BadgeService Interface is responsible for managing badges within the system. The **createBadge** method initiates the creation of badges, requiring a title and a list of variables as rules. The interface further supports awarding badges to students through the **awardBadge** method.

**EvaluationService Interface:**

EvaluationService Interface focuses on evaluating students and teams, both automatically and manually. The **evaluate** methods take parameters such as the evaluation tool, student or team, and educator. The interface plays a crucial role in assessing performances and assigning appropriate evaluations.

These interfaces collectively form the core functionality of the CodeKala system, covering user management, communication, evaluation, and tournament/battle orchestration.

## Deployement view

Immagine che contiene testo, schermata, Rettangolo, Carattere

Descrizione generata automaticamente

The above deployment diagram shows the architecture of the system in its distributed nature. The nodes represent hardware or software execution environments and the middleware connecting them. The colors used in the diagram reflect the fact that the system is developed as a three-tier architecture (as described in the section 2.1). In fact, the nodes represented in yellow are part of the client tier, the one in green composes the application logic tier and the one in blue the data tier. Furthermore, it is shown a node representing the external services, which reside on external servers and are equipped with APIs that allow the communication with the system. In addition, it is reported the load balancer, which is important to manage the concurrent accesses of several clients to CKB web application. Finally, it can be observed that the cardinalities of the associations between the nodes are chosen accordingly to the fact that the functionalities of the application logic and data tiers have to be replicated on multiple devices in order to face possible failures and prevent information losses; in particular, it was deemed sufficient to consider three servers and two databases that compose the system, but in future implementations of the CKB system it could be considered necessary to integrate some additional devices.

## Runtime view

In this section all the sequence diagrams that have been made in the RASD document will be expanded specifying which services of the application server operate to provide such functionalities. The entire communication between the CKB application and the services is deployed through the services’ methods specified in the UML diagram in 2.3. Whilst, for what concerns methods about the communication between user and CKB application, they are generic and will be provided through specific elements in the user interfaces. In particular, from the side of the client: in order to upload some data, it is used a method upload(some parameters), in order to request a specific view, it is used a method get(specific view), in order to choose some parameters for filtering, it is used a method choose(some parameters) and so on. While, from the side of the application: in order to show a view it is used a method show(specific view), in order to show a success or error message it is used a method show(Message) and so on. In some situations the order of actions of these sequence diagrams differs from the original one in the RASD document. This choice has been made in order to be coherent with the specific methods (explicated only in the previous chapter) that have to be used.

## 2.5 Component Interfaces

Immagine che contiene testo, schermata, Carattere, design

Descrizione generata automaticamenteThe following diagram represents the dependencies among the interfaces of the system. These dependencies are justified by the usage relations defined in the previous UML diagram and are reported here to make them more clear.

## 2.6 Selected architectural styles and patterns

**Three tier architecture:**

As already mentioned in the section 2.1, the type of architecture chosen to implement the CKB system is a three tier architecture. This architecture requires the division of the system into three logical and physical tiers: the presentation tier, the application tier and the data tier. The first has the scope of delivering information to the users of the application and, at the same time, retrieving from them information that have to be processed to provide the functionalities of the system. The second correspond to the part of the system in which reside the main logic of the application; it has the scope of processing all the data retrieved both from the users and the data tier. The application tier is also responsible of managing (i.e. inserting, removing and updating) the data present in the data tier, communicating with it by means of API calls. Finally, the data tier is where all the necessary data are stored. In a three tier architecture, it is not possible for the presentation tier to interact directly with the data tier, in fact, all communication have to go through the application tier. The main benefit of such type of architecture, which is also the reason why it has been chosen for the CKB system, is that the three tiers can be developed separately and run on different infrastructures. In such way, it will be also easier to update and extend the system whenever it is needed, modifying only the appropriate tier without any undesirable side effect on the other two tiers.

**Component based development:**

As it is highlighted in the previous sections, the development of the system is structured in com- ponents and sub-components. Therefore, it has been put into practice a component based devel- opment, which brings a great number of benefits both in short and long terms. In a system with a great number of functionalities, a component-based approach can be useful in order to develop a reusable software system, defining, implementing and composing loosely coupled independent components.

**Relational DBMS:**

A DBMS is a software system that allows the storage and management of data in a database. In the development of the CKB system it has been chosen a relational DBMS, which is a type of management system that stores data in a row-based table structure, in which the data of different tables are related. Furthermore, the DBMS implements some functions that deal with the security, accuracy, integrity and consistency of the data.

**Model View Controller (MVC):**

It is a software design pattern that requires the presence of three main logical components: the model, the view and the controller. The model has the scope of managing the data of the database and it corresponds, in the CKB system, to the classes defined in the class diagram (section 2.2.1). The View deals with the presentation of the information to the users and resides in the presentation tier. The Controller allows the communication between view and model and contains the services (defined in section 2.2.2) that provide all the needed functionalities. The main reason why it has been chosen this pattern is that it leads the software to be easily modifiable and its components widely reusable.

## 2.7 Other Design Decisions

### 2.7.1 Used Algotithms

**Mandatory Automated Evaluation**

The automated evaluation process in the CodeKataBattle (CKB) platform is designed to objectively assess the performance of student teams based on functional aspects, timeliness, and the quality level of their source code. The following algorithm outlines the steps involved in the automated evaluation:

1. **Functional Aspects (Test Cases):**

* **Objective:** Evaluate the functionality of the implemented solution.
* **Algorithm:**
  + Calculate the percentage of passed test cases out of all test cases.
  + Score= (Number of Passed Test Cases/Total Number of Test Cases)\*100

2. **Timeliness:**

* **Objective:** Encourage timely submissions by penalizing delays.
* **Algorithm:**
  + Calculate the time passed between the registration deadline and the last commit.
  + Assign a score inversely proportional to the time passed, with a maximum score for immediate submissions.

3. **Quality Level of Sources:**

* **Objective:** Assess the quality of the source code in terms of security, reliability, and maintainability.
* **Algorithm:**
  + Use static analysis tools selected by educators to evaluate various aspects.
  + Score based on the quality level of sources as determined by static analysis.

4. **Overall Automated Score:**

* **Objective:** Combine individual scores from functional aspects, timeliness, and source code quality.
* **Algorithm:**
  + Weight each score based on educator preferences.
  + Overall Automated Score=x1\*Functional Aspects Score + x2\*Timeliness Score + x3×Quality Level Score
  + Weights x1, x2, x3 are based on the relative importance of each aspect.