CodeKataBattle project by Russo Mario
and Picone Paolo

POLITECNICO
MILANO 1863

Design Document

Deliverable:	DD
Title:	Design Document
Authors:	Picone Paolo and Russo Mario
Version:	1.0
Date:	28-December-2023
Download page:	https://github.com/piconepaolo/PiconeRusso
Copyright:	Copyright © 2023, Picone Paolo and Russo Mario – All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

The motivation behind the existence of the CodeKataBattle platform is to provide students with a dedicated platform for practicing their programming skills. The platform aims to create a competitive environment where teams of students can participate in programming tournaments and solve programming challenges.

By offering a platform specifically designed for programming practice, CodeKataBattle aims to provide students with a structured and engaging way to improve their coding abilities. The competitive nature of the platform adds an extra layer of motivation and excitement, encouraging students to push their limits and strive for excellence.

Overall, the motivations behind the existence of the CodeKataBattle platform are to create a dedicated space for programming practice, foster healthy competition among students, and provide a means for tracking and improving programming skills.

1.2 Scope

...

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **Educator**: a person who is responsible for creating and managing tournaments and battles
- **Student**: a person who is participating in tournaments and battles
- **Team**: a group of students that participate in a battle
- **Battle**: a programming challenge that takes place in a tournament
- **Code Kata**: a programming challenge that defines a battle
- **GitHub**: a web-based hosting service for version control using Git
- **GitHub Actions**: a feature of GitHub that allows automating tasks
- **GitHub repository**: a storage space where the project files are stored
- **GitHub fork**: a copy of a repository
- **Platform**: The interface that allows the interaction between the user and the system
- **System**: The software that implements the functionalities of the platform

1.3.2 Acronyms

- **CKB**: CodeKataBattle

1.3.3 Abbreviations

- **WP**: World Phenomena
- **SP**: Shared Phenomena
- **G**: Goal
- **R**: Requirement
- **UC**: Use Case
- **UI**: User Interface
- **API**: Application Programming Interface

1.4 Revision history

1.5 Reference documents

This document is based on:

- The specification of the RASD assignment of the Software Engineering 2 course
- The slides of the Software Engineering 2 course

1.6 Document structure

This document is structured as follows:

- **Introduction**: it provides a general description of the product, its purpose and the goals that the project aims to achieve. It also contains the scope of the product, the phenomena that the product will interact with and the shared phenomena between the product and the world. Finally, it contains the definitions, acronyms and abbreviations used in the document.
- **Overall Description**: it's a high level description of the product. It contains the product perspective, the product functions, the user characteristics, the constraints, the assumptions and the dependencies of the product.
- **Specific Requirements**: it contains the functional and non-functional requirements of the product. It also contains the use cases, the sequence diagrams of the most important use cases and the external interfaces.
- **Formal Analysis**: it contains the Alloy model of the product.
- **Effort Spent**: it contains the number of hours spent by each member of the group to redact this document.
- **References**: it contains the list of the documents used to redact this document.

2 Architectural Design

2.1 Overview: High level components and their interaction

The system is divided into three main layers: presentation layer, application layer and data layer. The presentation layer is the interface between the user and the system. It is responsible for the presentation of the data and the interaction with the user. The application layer is the core of the CKB Platform. It is responsible for the business logic and the communication between the presentation layer and the data layer. The data layer is responsible for the storage of the data. It is the interface between the application layer and the database.

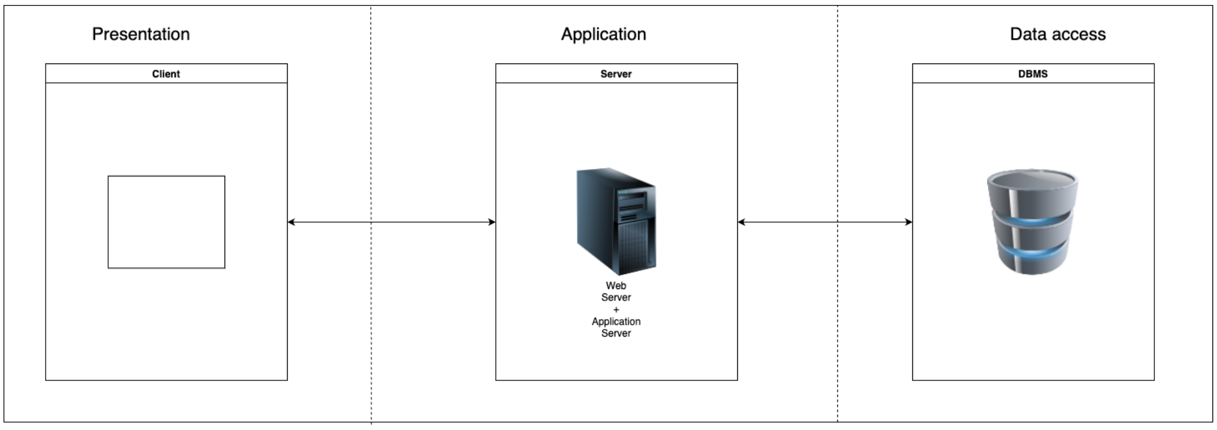

Figure 1: High level components and their interaction

The three-tier architecture was chosen for this type of system because of the several advantages it offers compared to other types of architectures. For that concerns scalability, it allows for easy scalability by separating the presentation, application, and data layers. Each layer can be scaled independently, allowing for better performance and resource utilization. For what concers modulatiry, it promotes the separation of concers by dividing the system into distinct layers. This makes it easier to develop, test, and maintain each layer separately, improving overall code quality and reusability. The sepatarion of concerns also improves the maintainability of the system by providing clear boundaries between layers it makes easier to understand and modify specific parts of the system without affecting other layers. The last advantage for which this architecture was chosen is flexibility. The three-tier provides flexibility in terms of technology choices. Each layer can be implemented using different technologies, allowing for the use of the most suitable tools and frameworks for each specific layer.

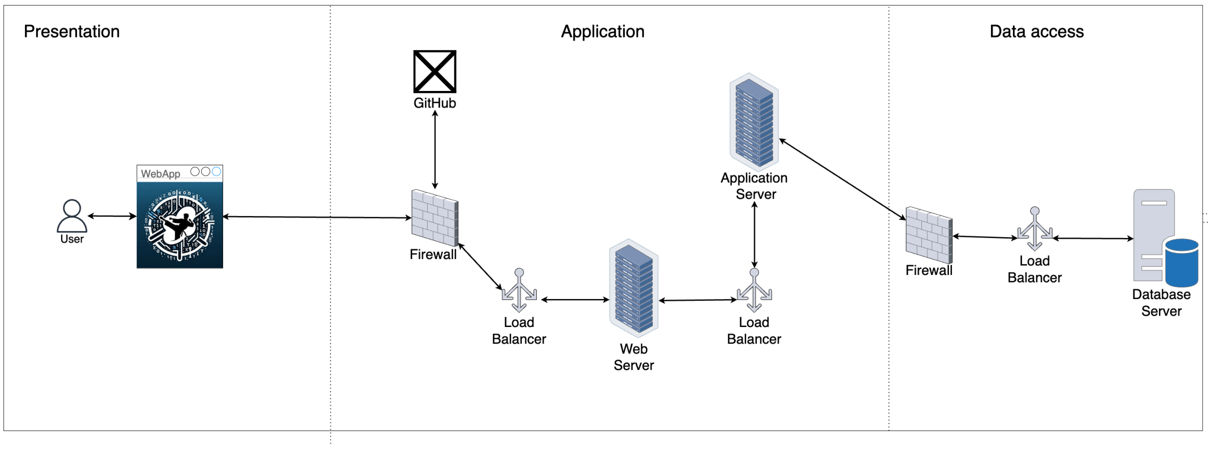

Figure 2: High level system with interactions between the components

In the figure Figure 2 is shown the high level system with the interactions between the components. At the forefront is the Presentation layer, where a user engages with the web application through a web browser. The web application depending on user interactions requests data from the web server. The interaction between the two is not direct since there is a firewall (for security reasons) and also a load balancer.

Moving inward, the Application layer serves as the system's operational core, where a network firewall establishes the first line of defense, safeguarding the internal processes. A load balancer stands right behind the firewall, directing incoming traffic to maintain system efficiency and reliability. This layer is further composed by an application server that executes the business logic, interfacing with databases or other external services as needed. Additionally, the presence of an upward arrow connecting the load balancer to GitHub to handle the evaluation trigger as well as the creation of the repositories. This is a very important feature of the system since it allows to automatically evaluate the students' submissions.

The final segment of the diagram is the Data Access layer, echoing the security and balance themes with its own firewall and load balancer, underscoring the system's commitment to secure data transactions. At the heart of this layer lies the database server, a robust storage solution that ensures data is efficiently stored, retrieved, and managed, completing the architecture's promise of a secure, scalable, and resilient web application environment.

2.2 Component view

In Figure 3 is shown the component diagram of the system. The system is divided into three main layers: presentation layer, application layer and data layer. The presentation layer is the interface between the user and the system and it is represented with the WebApp and Web Server Component. The application layer is the core of the CKB Platform and it is representend by the Application Server Subsystem. Finally the data layer is represented by the DBMS Component who is the only one that access the database.

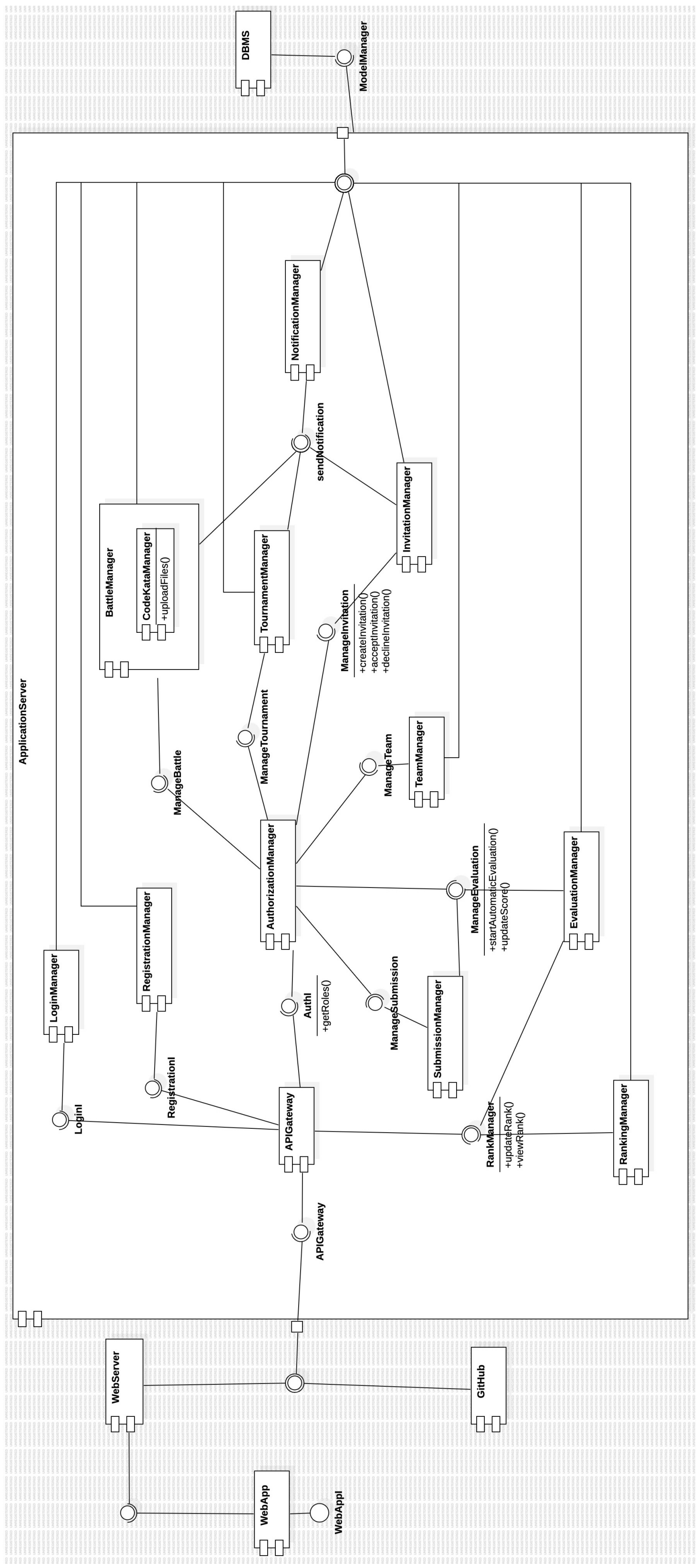

Figure 3: Component diagram

Regarding the presentation layer, it is represented by the following two components:

- **WebApp**: it is the web application that the user interacts with. It is responsible for the presentation of the data and the interaction with the user. Since it communicates only with the Web Server, it can be executed by any device that has a web browser.
- **Web Server**: it is the component that handles the requests from the WebApp and forwards them to the Application Server. It is responsible to direct the requests to the APIGateway. It is also responsible to forward responses from the Application Server to the WebApp.

Regarding the application layer, it is represented by the components inside the Application Server Subsystem:

- **APIGateway**: it is the component that handles the incoming requests to the Application Server. It is responsible to direct the requests to the correct component inside the Application Server Subsystem. It is also responsible to forward responses from the Application Server to the Web Server.
- **Login Manager**: it is the component that handles the login requests. It is responsible to check the credentials and to log the user in.
- **Registration Manager**: it is the component that handles the registration requests. It is responsible to create a new user.
- **Authorization Manager**: it is the component that handles the authorization requests. It is responsible to check if the user is authorized to perform the requested action. This is of paramount importance since only a user with Educator role can perform administrative actions, such as the creation of a new Tournament or Battle.
- **Evaluation Manager**: it is the component that handles the evaluation requests. It is responsible to evaluate the submissions of the students and to return the results. It performs the evaluation by running and testing the submissions provided by the students. It can be used also by the Educator to re-evaluate the submissions manually.
- **Ranking Manager**: it is the component that handles the rank of students in a Tournament.
- **Team Manager**: it is the component that handles the creation of teams and the assignment of students to teams. It is also responsible to guarantee that the restrictions on the number of students per team, set by the Educator, are respected.
- **Submission Manager**: it is the component that handles the submission of the students. The submission is created when the GitHub Action is triggered by the push of the student in the Team repository. It interacts with the Evaluation Manager to get the evaluation of the submission.
- **Invitation Manager**: it is the component that handles the invitation of both Students and Educator. In the case of the Students it is responsible to manage the invitation to join a Team. In the case of the Educator it is responsible to manage the invitation to join a Tournament.
- **Notification Manager**: it is the component that handles the notification requests. It is responsible to send notifications to the users. Such notifications are sent respecting the conditions previously specified in the RASD document.
- **Battle Manager**: it is the component that handles the creation and the management of Battles by the Educator. It is a complex component since it embodies also the CodeKata Manager. Since it contains the CodeKata Manager, it is responsible to handle the CodeKata files uploaded by the Educator. It interacts with the GitHub Component to create the repositories for the Battles. It also handles the different Battle phases, by guaranteeing the deadline restrictions set by the Educator.

- **Tournament Manager**: it is the component that handles the creation and the management of Tournaments by the Educator. It is responsible to handle the different Tournament phases, by guaranteeing the deadline restrictions set by the Educator.

Regarding the data layer, it is represented by the following component:

- **DBMS**: it is the component that handles the access to the database. It is responsible to store and retrieve the data from the database. It is the only component that can access the database.

2.3 Deployment view

In this section is shown the deployment diagram of the system. The diagram is divided into three main tier: the presentation tier, the application tier and the data tier.

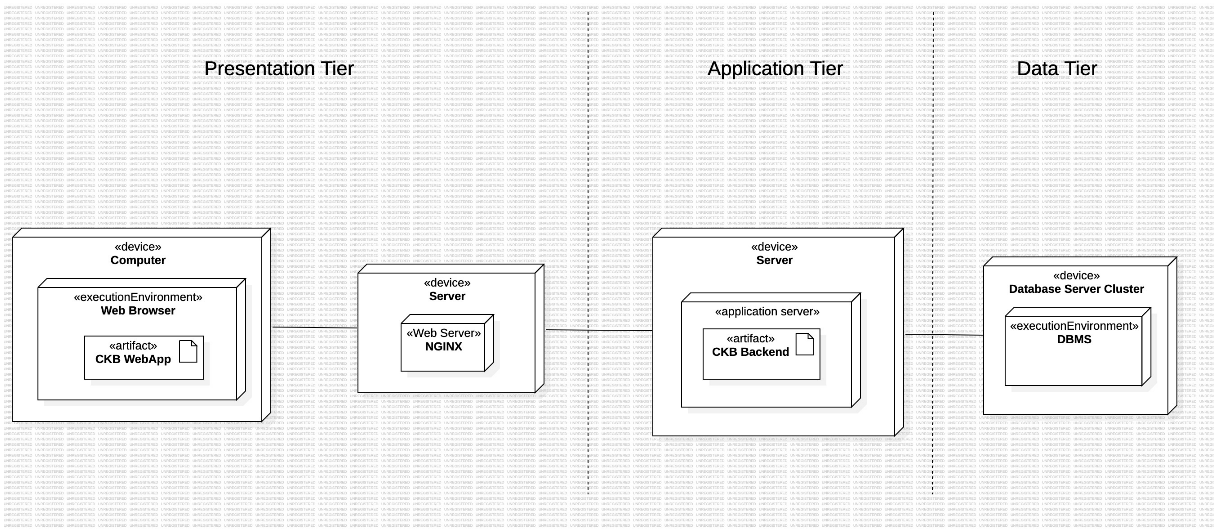

Figure 4: Deployment diagram

In figure Figure 4 is shown the deployment diagram of the system. Regarding the presentation tier, it is represented by the following components:

- **Computer**: it is a normal computer used by the user. It is the device that the user uses to interact with the CKB WebApp. It has no particular requirements since it only needs a web browser to access the WebApp.
- **Web Browser**: it is the software that the user uses to interact with the CKB WebApp. The CKB platform has to support the most common web browsers, such as Google Chrome, Mozilla Firefox, Microsoft Edge and Apple Safari.
- **CKB WebApp**: it is the web application that the user interacts with and it is used within the web browser. It is responsible for the presentation of the data and the interaction with the user.
- **Web Server**: it is a server that hosts a NGINX web server instance. The choice of NGINX is due to its high performance, low memory usage and high availability.

Regarding the application tier, it is represented by the following components:

- **Application Server**: it is a server that hosts a CKB Backend application instance. It receives the requests from the Web Server and forwards it to the CKB Backend. At the same time it provides responses to the Web Server. It is also responsible to communicate with the DBMS to retrieve and store data.

- **CKB Backend**: it is the backend application of the CKB Platform. It handle the entire business logic of the application. It is responsible to process the forwarded requests and to provide responses to the Application Server.

Regarding the data tier, it is represented by the following components:

- **Database Server Cluster**: it is a cluster of servers that hosts a DBMS instance. The choice of a cluster is due to the fact that it provides high availability and scalability. It is also responsible to guarantee the security of the data. It process the database specific requests sent by the Application Server.
- **DBMS**: it is the software that manages the database. It is responsible to store and retrieve the data from the database. It is the only component that can access the database.

2.4 Component interfaces

Interface LoginManager:

- `Session login(String username, String password);`
- `void logout(Session session);`

Interface RegistrationManager:

- `boolean register(UserDetails userDetails);`

Interface AuthorizationManager:

- `List<Role> getRoles(User user);`
  Get the roles of a user. The roles are used to determine if a user is authorized to perform a specific action.

Interface BattleManager:

- `Battle createBattle(BattleDetails details);`
  Create a new Battle. The BattleDetails contains all the information needed to create a new Battle.
- `boolean uploadFiles(Battle battle, CodeKata files);`
  Upload the files of the CodeKata. The files must contain the tests and the automation scripts of the CodeKata.
- `boolean updateBattle(Battle battle, BattleDetails details);` Updates the details of a Battle, such as the different stages and the deadlines.
- `List<Team> getTeams(Battle battle);`

Interface TournamentManager:

- `Tournament createTournament(TournamentDetails details);`
  Create a new Tournament. The TournamentDetails contains all the information needed to create a new Tournament.
- `boolean updateTournament(Tournament tournament, TournamentDetails details);`
  Updates the details of a Tournament, such as the different stages and the deadlines.

Interface InvitationManager:

- `boolean sendInvitation(User user, Invitation invitation);`
  Send an invitation to a user. The invitation can be to join a Tournament, if the user is an Educator, or to join a Team, if the user is a Student.
- `boolean acceptInvitation(Invitation invitation);`
  Accept an invitation sent by another user.
- `boolean declineInvitation(Invitation invitation);`
  Decline an invitation sent by another user.

Interface RankingManager:

- `void updateRank(Student student, Tournament tournament, int newRank);`
  Update the rank of a Student in a Tournament. This method can only be used by users with Educator Role.
- `int viewRank(Student student, Tournament tournament);`
  View the rank of a Student in a Tournament.

Interface SubmissionManager:

- `SubmissionResult submit(Item item);`
  Submit a solution to a Battle. The Item contains the code of the solution. The SubmissionResult contains the evaluation of the submission.
- `Item getSubmission(int submissionId);`
  Get the code of a submission given its id.
- `SubmissionResult getSubmissionResult(int submissionId);`
  Get the evaluation of a submission given its id.

Interface NotificationManager:

- `void notify(User user, Notification notification);`
  Send a notification to a user.

Interface ModelManager:

- `boolean saveModel(DataModel model);`
  Save a DataModel in the database.
- `DataModel getModel(int modelId);`
  Get a DataModel from the database given its id.
- `boolean updateModel(DataModel model);`
  Update a DataModel in the database.
- `boolean deleteModel(DataModel model);`
  Delete a DataModel from the database.

Interface TeamManager:

- `Team createTeam(TeamDetails details);`
  Create a new Team. The TeamDetails contains all the information needed to create a new Team.
- `boolean updateTeam(Team team, TeamDetails details);`
  Updates the details of a Team, such as the name and the members.
- `boolean addMember(Team team, Student student);`
  Add a Student to a Team.

2.5 Runtime view

2.6 Selected architectural styles and patterns

2.7 Other design decisions

3 User Interface Design

4 Requirement Traceability

5 Implementation, Integration and Test Plan

5.1 Overview

5.2 Implementation plan

5.2.1 Feature identification

5.2.2 Component Integration and Testing

5.3 System testing

5.4 Additional specifications on testing

6 Effort Spent

This section provides an estimation of the effort spent by each member of the group to redact this document. The time for each section includes the time spent to write, to discuss and to review the document itself.

Picone Paolo

Section	**Hours**
1	6
2	13
3	23
4	12

Russo Mario

Section	**Hours**
1	6
2	14
3	22
4	13

References

- Document written using LaTeX and Visual Studio Code
- Diagrams created using StarUML 6.0.1
- Alloy code created using Alloy Analyzer 6.1.0
- Mockups created using Figma