POLITECNICO
MILANO 1863

CodeKataBattle project by Russo Mario
and Picone Paolo

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

Table of Contents **3**

List of Figures **4**

List of Tables **4**

1 Introduction **5**
1.1 Purpose 5
1.2 Scope 5
1.3 Definitions, Acronyms, Abbreviations 5
1.3.1 Definitions 5
1.3.2 Acronyms 5
1.3.3 Abbreviations 6
1.4 Revision history 6
1.5 Reference documents 6
1.6 Document structure 6

2 Architectural Design **7**
2.1 Overview: High level components and their interaction 7
2.2 Component view 10
2.3 Deployment view 11
2.4 Component interfaces 11
2.5 Runtime view 11
2.6 Selected architectural styles and patterns 11
2.7 Other design decisions 11

3 User Interface Design **12**

4 Requirement Traceability **13**

5 Implementation, Integration and Test Plan **14**
5.1 Overview 14
5.2 Implementation plan 14
5.2.1 Feature identification 14
5.2.2 Component Integration and Testing 14
5.3 System testing 14
5.4 Additional specifications on testing 14

6 Effort Spent **15**

References **16**

List of Figures

1 High level components and their interaction 7
2 High level system with interactions between the components 7
3 Component diagram 10

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.

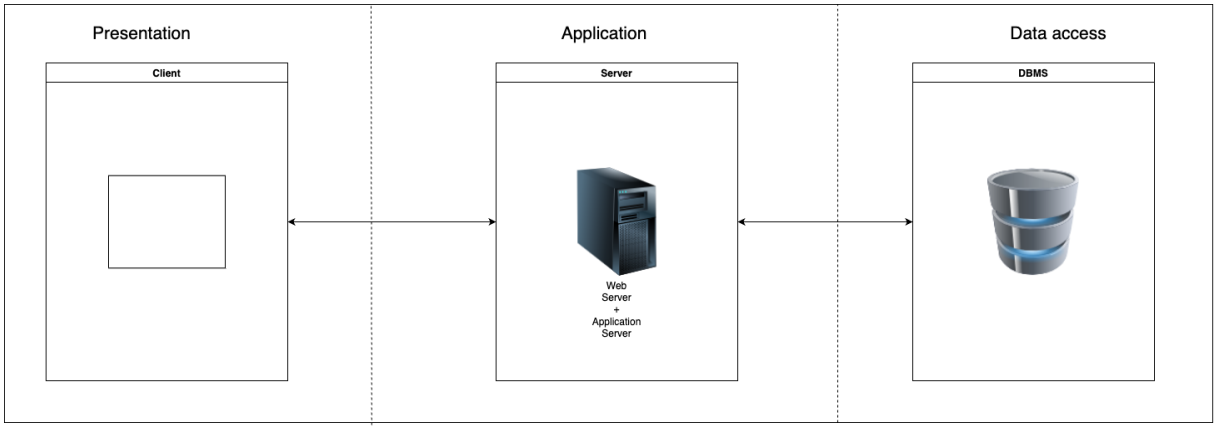

Figure 1: High level components and their interaction

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.

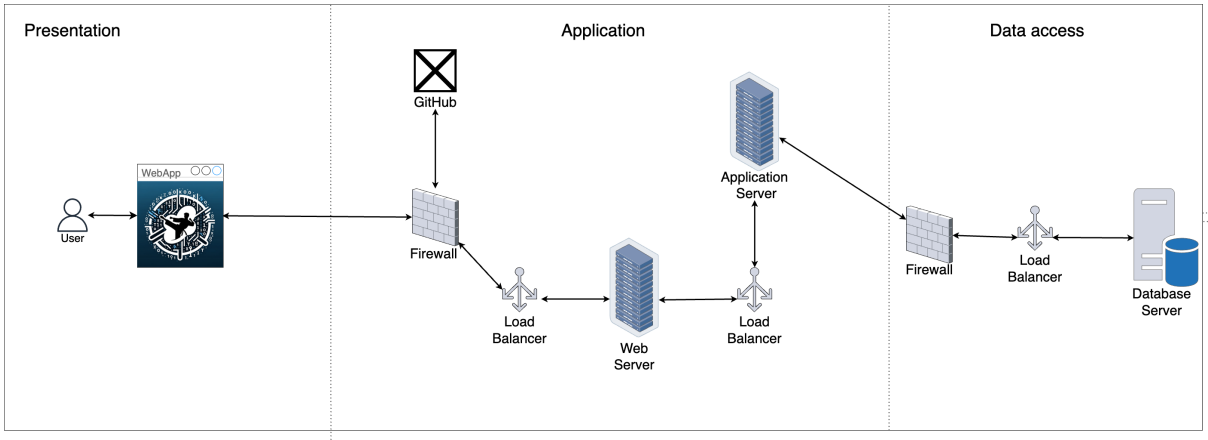

Figure 2: High level system with interactions between the components

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

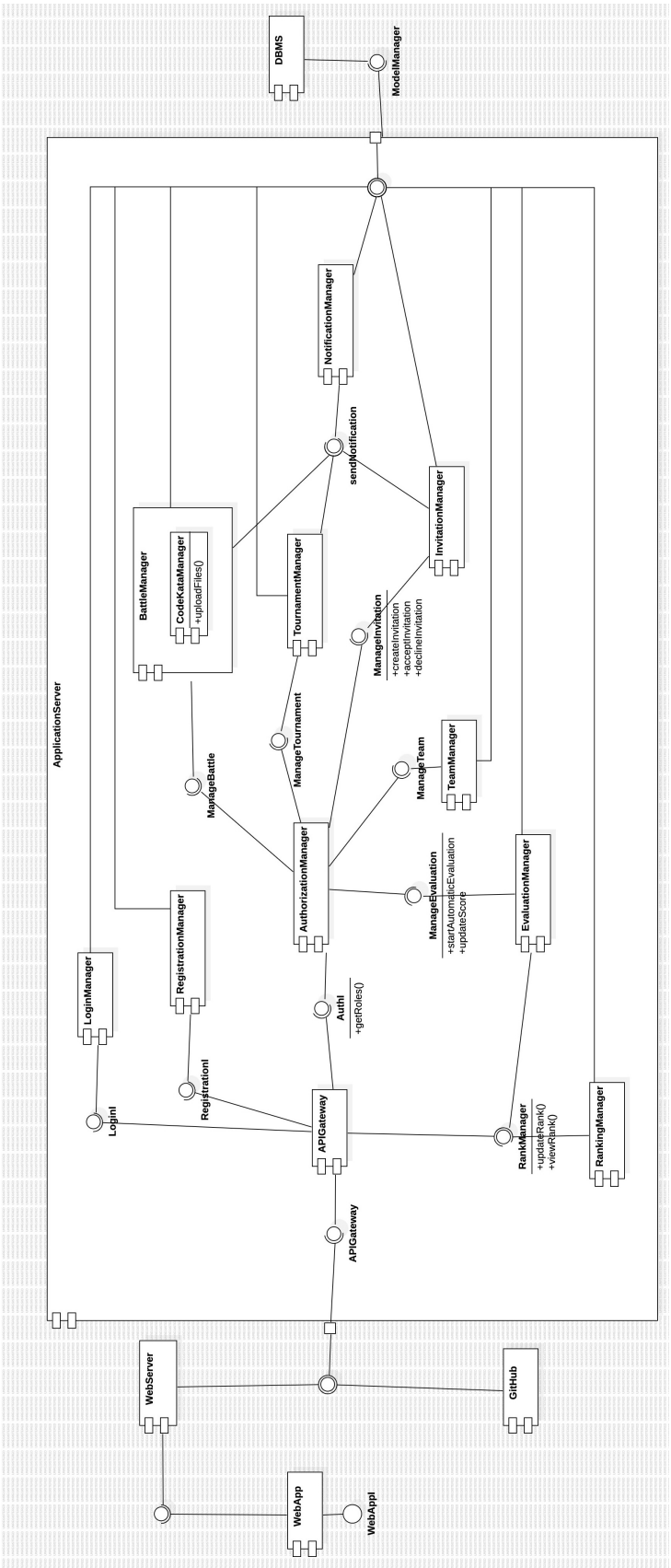

Figure 3: Component diagram

2.3 Deployment view

2.4 Component interfaces

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

