AY 2023/2024

POLITECNICO
MILANO 1863

DD: Design Document

Tommaso Capacci Gabriele Ginestroni

Professor
Elisabetta Di Nitto

Version 1
December 13, 2023

Contents

1 Introduction

1.1 Scope

Traditional software programming education often lacks of hands-on experience and continuous evaluation. CodeKataBattle addresses these issues by providing a platform for competitive programming challenges which promotes teamwork and emphasizes the test-first approach in software development. It allows students to apply theoretical knowledge in tournaments with an automated scoring system. Instructors benefit from closer mentorship through code reviews and manual evaluation, creating a cycle of learning through practice, feedback, and community engagement.

CodeKataBattle's platform will employ a microservices architecture, emphasizing the decomposition of the system into small, independently deployable services. Each microservice will be dedicated to a specific business capability, promoting modular development and ease of maintenance.The architecture facilitates scalability, allowing individual services to be scaled independently based on demand. Other important design choices include:

- **Service Discovery**: A service registry will be used to allow services to locate each other without prior knowledge of their location. This enables efficient communication between services by providing up-to-date information. Service discovery enhances fault tolerance and load balancing, contributing to the overall reliability of the system.
- **API Gateway**: An API gateway will be used to provide a single entry point for clients to access the system. This simplifies the client interface by abstracting the underlying microservices and provides a centralized location for authentication and authorization. The API Gateway enhances security measures, ensuring controlled and secure access to the microservices.
- **Hybrid Communication Framework**: While most services exploit synchronous communication via REST api calls, an event-driven communication framework is also used to facilitate communication between specific microservices, allowing them to be loosely coupled and promoting both modularity and scalability. This framework enhances the

reliability of these services by providing a mechanism for asynchronous communication between them (i.e. queues).

- **Data Management Strategies**: The system employs tailored data management strategies, utilizing databases suitable for microservices. Both relational and NoSQL databases are considered for each specific service in order to provide flexibility and scalability.

Incorporating these key properties into the CodeKataBattle system should provide a robust and scalable architecture, ensuring modularity, responsiveness, security, reliability, and effective data management.

1.2 Definitions, Acronyms, Abbreviations

TODO: remove unused ones

1.2.1 Definitions

- **User:** anyone that has registered to the platform
- **Student:** the first kind of users and, basically, the people this product is designed for. Their objective is to submit solutions to battles
- **Team:** students can decide to group up and form a team to partecipate to a battle. The score assigned to the submission of a team will be assigned also to each one of its members
- **Educator:** the second kind of users. They create tournaments, set-up battles, and eventually, evaluate the solutions that teams of students have submitted during the challenge
- **Tournament:** collection of coding exercises (battles) about specific topics of a subject. Interested students can subscribe to it and participate to its battles as soon as they are published
- **Code Kata Battle (or battle):** the atomic unit of a tournament. Usually students are asked to implement an algorithm or to develope a simple project that solves the task. Each battle belongs to a specific tournament: students submitting solutions for a battle will obtain a score that will be used to compute both the team's rank for the battle and the members' tournament rank

- **Code Kata:** description and software project necessary for the battle, including test cases and build automation scripts. These are uploaded by the educator at battle creation time
- **Tournament collaborator:** other educator that is added by the tournament creator to help him in the management of the tournament. He can create battles and evaluate their submissions
- **GitHub:** web-based hosting service for version control, mostly used for computer code. It offers both distributed version control and source code management functionalities
- **GitHub repository:** a repository is a storage space where some project files are stored. It can be either public or private. In the context of the platform, each battle is associated with a GitHub repository that is created by the system and shared with the teams
- **GitHub collaborator:** person who is granted access to a GitHub repository with write permission
- **GitHub Actions:** GitHub feature that allows to automate tasks directly on GitHub, such as building and testing code, or deploying applications. In the context of the platform, GitHub Actions is used to automatically notify the system when a new submission is pushed to the repository of a team
- **Test cases:** each battle is associated with a set of test cases, which are input-output value pairs that describe the correct behavior of the ideal solution
- **Static analysis:** is the analysis of programs performed without executing them, usually achieved by applying formal methods directly to the source code. In the context of the platform this kind of analysis is used to extract additional information about the level of security, reliability and maintainability of a battle submission
- **Functional analysis:** measures the correctness of a solution in terms of passed test cases
- **Timeliness:** measures the time passed between the start of the battle and the last commit of the submission

- **Score:** to each solution is assigned a score which is computed taking into account timeliness, functional and static analysis and, eventually, manual score assigned by the educator that created the challenge. The score is a natural number between 0 and 100 (the higher the better)
- **Rank:** during a battle, students can visualize the ranking of teams taking part to the battle. Moreover, at the end of each battle, the platform updates the personal tournament score of each student. Specifically, the score is computed as the sum of all the battles scores received in that tournament. This overall score is used to fill out a ranking of all the students participating to the tournament which is accessible by any time and by any user subscribed to the platform
- **Notification:** it's an email alert that is sent to users to inform them that a certain event occurred such as the creation of a new tournament and battle or the publication of the final rank of a battle

1.2.2 Acronyms

- **DD:** Design Document
- **RASD:** Requirements Analysis and Specification Document
- **CKB:** Code Kata Battles
- **API:** Application Programming Interface
- **UML:** Unified Modeling Language
- **HTML:** HyperText Markup Language
- **CSS:** Cascading Style Sheets
- **JSON:** JavaScript Object Notation
- **OS:** Operating System
- **REST:** REpresentational State Transfer
- **URL:** Uniform Resource Locator
- **HTTPS:** HyperText Transfer Protocol Secure

1.2.3 Abbreviations

- **Gn:** Goal number “n”
- **Dn:** Domain Assumption number “n”
- **Rn:** Requirement number “n”
- **UCn:** Use Case number “n”

1.3 Revision History

TODO

1.4 Reference Documents

- Specification document: "Assignment RDD AY 2023-2024"
- DD reference template: "04e.QualitiesAndCreatingDD.pdf"
- UML official specification https://www.omg.org/spec/UML
- GitHub API official documentation: https://docs.github.com/en/rest/guides/getting-started-with-the-rest-api?apiVersion=2022-11-28

1.5 Document Structure

TODO: check if this is enough

- **Section 1: Introduction**
  In this section a general description of the document and the system to be developed is provided, also including a glossary of terms used and a list of reference documents.
- **Section 2: Architectural Design**
  Here the high-level structure of the software is outlined. This includes the identification of major components, their interactions, and the overall flow of data within the system. Dependencies on external factors or third-party integrations are also detailed, offering a comprehensive view of the software’s architecture.

- **Section 3: User Interface Design**
  Focused on the end-user experience, this section describes the layout, interactivity, and visual elements of the software's user interface. It includes mockups of the main pages of the web application.

- **Section 4: Requirements Traceability**
  Here the relation between software requirements and design elements is highlighted. This is achieved through the use of a traceability matrix.

- **Section 5: Implementation, Integration and Test Plan**
  This section is a comprehensive guide that covers the main aspects of the software development lifecycle. It outlines the details of implementation and integration plan, as well as the testing strategy.

- **Section 6: Effort spent**
  The sixth and last chapter contains the time spent by each contributor of this document.

2 Architectural Design

2.1 Overview

The following diagram represents the high level architecture of the system, including the external entities that will interact with it.

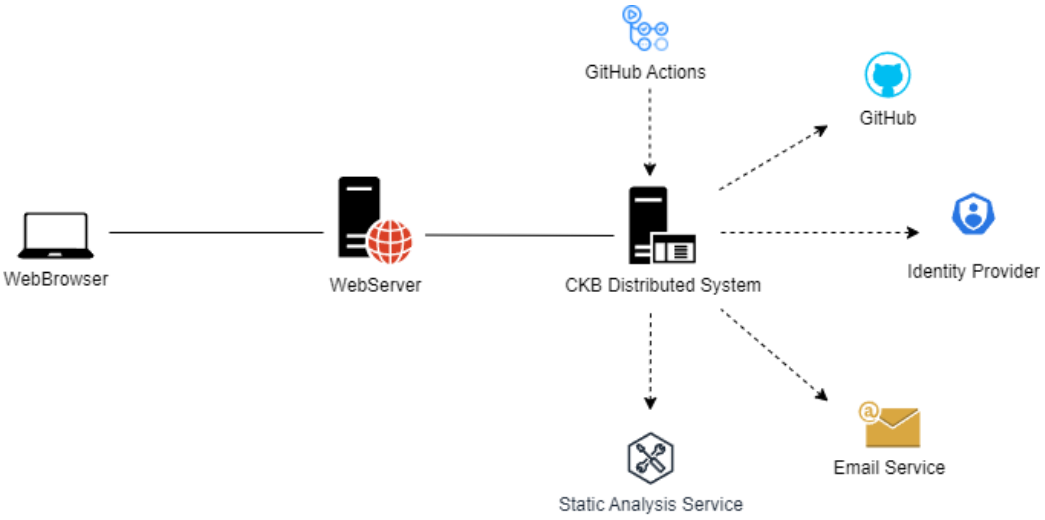

Figure 1: CKB system diagram

The main elements contained in Figure 1 are:

- **WebBrowser**: used by students and educators to access the system functionalities through the web application.
- **WebServer**: its functions are:
  - Serving static assets (HTML, CSS, JavaScript) to the client, necessary for handling the initial rendering of the UI.
  - Managing the client-side application and routing.
  - Generating requests directed to the backend system.
- **CKB Distributed System**: the distributed system composed of multiple microservices that implement the core functionalities of the system. Its in charge of the data management, application and integration logic of the whole CKB platform.

- **External Entities**: the CKB Distributed System must be able to integrate with external actors to accomplish its functionalities. The arrow in the diagram highlights the direction of the interaction between the system and the external entities.

2.2 Component view

The following component diagram highlights the main components of the system and their interaction with external entities and services. In the diagram, the components have been organized to highlight the logical grouping of the system elements.
The WebApplication component represents the presentation layer of the system, being the only entry point for the users. The application and integration logic are represented together due to their tight interaction, while the data layer contains the databases accessed by the respective microservices. Different colors are used to highlight components that share similar roles in the system.
Orange components represent the system's microservices. Some complex microservices have been further decomposed into subcomponents, for a more fine grained representation.
Yellow components represent the model of the database accessed by its microservice. The model offers to the microservice an abstraction of the database, allowing it to access the data without knowing the underlying database implementation technology.
The violet has been used to highlight components that cover an important role in the integration between some of the main microservices of the system. Specifically, it has been used for the queues subcomponents, which are used to implement the asynchronous and concurrent communication between specific microservices. Some microservices have an important role in the integration with external entities as well, but have been depicted with their orange color used for microservices. This aspect will be clarified in the detailed description of the components that follows the diagram.
Red components represent the external services that interact with the system.
Finally, the green color has been used to highlight the databases components that are used to store the data of the system.
It's important to notice that for the sake of simplicity and readability, the interface offered by the ServiceRegistry has been depicted with some dotted

arrows that connect all the microservices to it.

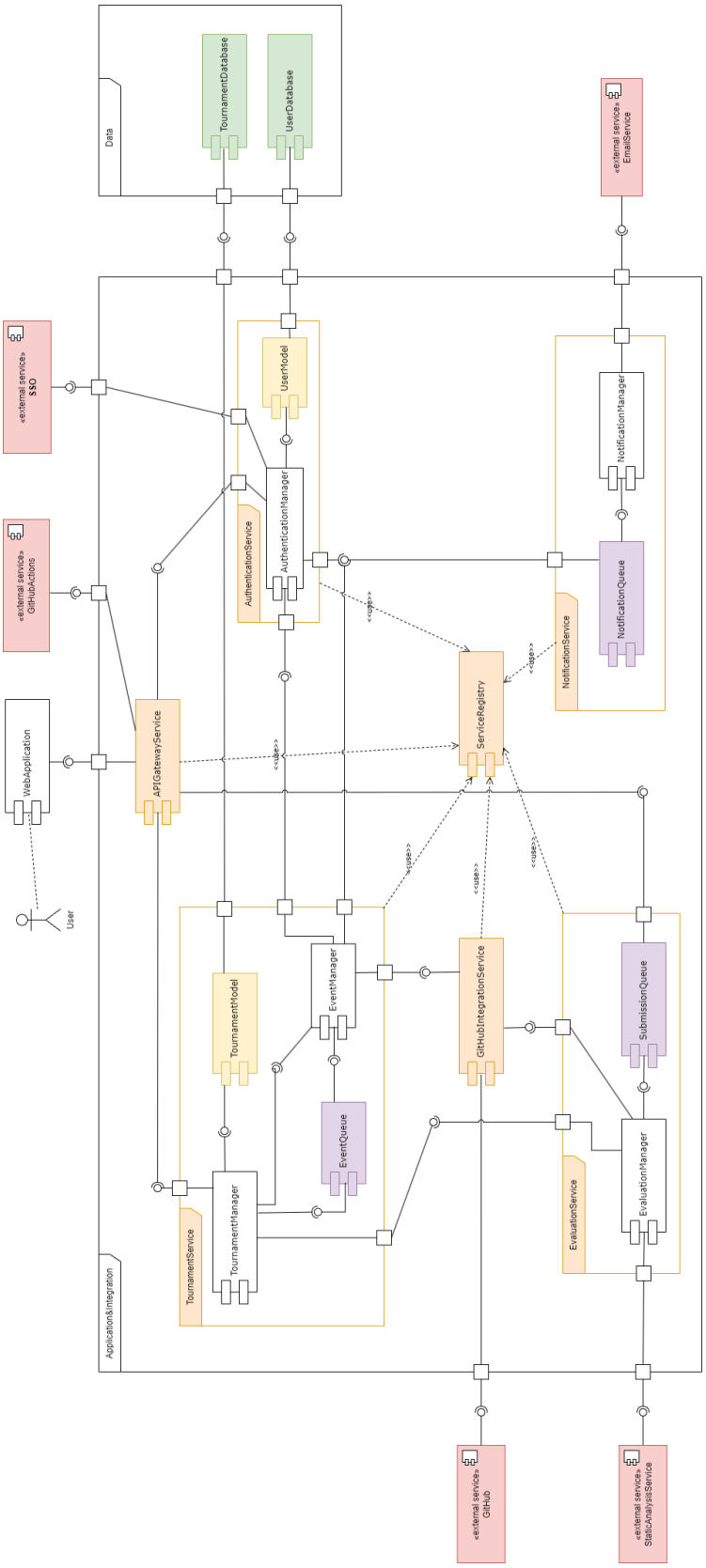

Figure 2: Component diagram

The components in Figure 2 are :

- **WebApplication**: the web app to which users of the CKB platform (students and educators) connect through a modern web browser. It is the front end of the system, and thanks to the interface offered by the **APIGatewayService**, allows users to manage and access the most important aspects of tournaments and battles
- **APIGatewayService**: this component is the microservice that exposes the REST API used by the WebApplication. Indeed, it allows the implementation of the main functionalities needed by the users of the web application by orchestrating the microservices. It also offers a REST API, used by the GitHub Actions Service, to notify a submission of a student on the Github repository. It's responsible for the following functionalities:
  - Acts as a single entry point for client requests.
  - Handles authentication, authorization.
  - Routes requests to the appropriate microservices that provide the required business functionality.
  - Aggregates responses from multiple microservices if needed.
  - Load balancing and API rate limiting.
- **AuthenticationService**: this component is the microservice that handles the authentication and authorization of the users of the system. It is responsible also for all the data related to the users of the system, such as their personal information and their roles. It includes the following subcomponents:
  - **AuthenticationManager**: implements the main logical functionalities of the AuthenticationService, exposing APIs used by other microservices to authenticate users and to retrieve their information.
  - **UserModel**: represents the model of the database used by the AuthenticationService to store the data related to the users of the system.
- **TournamentService**: this component is the microservice that implements most of the functionalities needed by the users. It handles:

  - Management of tournaments and battles: it allows the creation of battle and tournaments, enrollment of students to tournaments and battles.
  - Management of events regarding tournaments and battles.
  - Management of the ranking of the students.
  - Management of data related to tournaments, battles and submissions
  - Manual evaluation of the submissions of the students.

  It is composed of the following subcomponents:

  - **TournamentManager**: implements the main logical functionalities of the TournamentService, exposing APIs used by other microservices to manage tournaments and battles and their data.
  - **TournamentModel**: represents the model of the database used by the TournamentService to store the data related to tournaments and battles.
  - **EventQueue**: implements the queue used by the TournamentService to manage the events related to tournaments and battles.
  - **EventManager**: periodically checks the EventQueue for events and processes them once the deadlines are reached.

- **GitHubIntegrationService**: this component is the microservice that handles the integration with GitHub. It is responsible for the following functionalities:

  - Creation of the GitHub repository of the battle.
  - Retrieval of the code of the submission from the GitHub repository.

- **EvaluationService**: this component is the microservice that handles the evaluation of the submissions of the students. It is responsible for the following functionalities:

  - Evaluation of the submissions, in terms of timeliness and functional analysis

  - Integration with external static code analysis tools to evaluate the quality of the code of the submissions.

  It is composed of the following subcomponents:

  - **EvaluationManager**: implements the main logical functionalities of the EvaluationService, periodically checking the EvaluationQueue for submissions to evaluate and processing them.
  - **EvaluationQueue**: queue that stores notifications about new pending submissions, appended by the GitHubActionsService through the REST API exposed by the APIGatewayService, yet to be evaluated.

- **NotificationService**: this component is the microservice that handles the notifications of the users of the system. It is responsible for the following functionalities:

  - Dispatch of confirmation email to new registered users.
  - Dispatch of email notifications to users in case of events related to tournaments and battles.

- **ServiceRegistry**: this component is the microservice that handles the registration of the microservices to the system. It offers to all the other microservices the following:

  - Registration of the microservices istances to the system.
  - Discovery of the microservices istances by the other microservices.
  - Availability check of the microservices istances, by receiving periodic heartbeats from them.

- **TournamentDatabase**: this component is the database used by the system to store the data related to tournaments and battles, including also scores and ranks.

- **UserDatabase**: this component is the database used by the system to store the data related to the users of the system, such as kind of user, email and usernames.

The Figure 2 contains also some external entities the system interacts with:

- **GitHub**: used by the system to retrieve the code of GitHub repositories and to create new repositories.
- **GitHubActions**: configured by the students on their GitHub repository to automatically notify the system when a new submission is pushed to the repository.
- **StaticAnalysisService**: used by the system to evaluate the quality of the code of the submissions.
- **EmailService**: used by the system to send emails to the users of the system.
- **SSO**: used by the system to offer the users the possibility to signup and login with their preferred identity provider.

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 Effort Spent

Name and Surname	Section 1	Section 2	Section 3	Section 4
Tommaso Capacci	10	10	10	10
Gabriele Ginestroni	10	10	10	10