AY 2023/2024

POLITECNICO
MILANO 1863

DD: Design Document

Tommaso Capacci Gabriele Ginestroni

Professor
Elisabetta Di Nitto

Version 1
December 9, 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.
- **Event-Driven COmmunication Framework**: An event-driven communication framework will be used to facilitate communication between microservices. This allows services to be loosely coupled, promoting modularity and scalability. The framework enhances the reliability of the system by providing a mechanism for asynchronous communication between services (i.e. queues).

- **Data Management Strategies**: The system employs tailored data management strategies, utilizing databases suitable for microservices. NoSQL databases are considered for specific services, providing flexibility and scalability. Eventual consistency is embraced for distributed data management, ensuring data isolation and autonomy for individual microservices.

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

1.2.1 Definitions

1.2.2 Acronyms

1.2.3 Abbreviations

1.3 Revision History

1.4 Reference Documents

1.5 Document Structure

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.

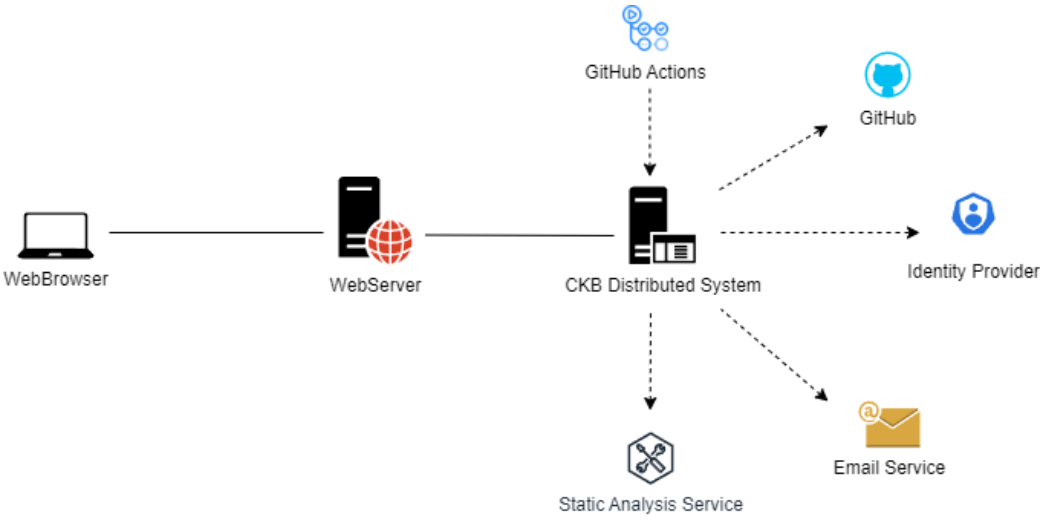

Figure 1: CKB system diagram

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 API requests to the API Gateway for dynamic data.
- **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.

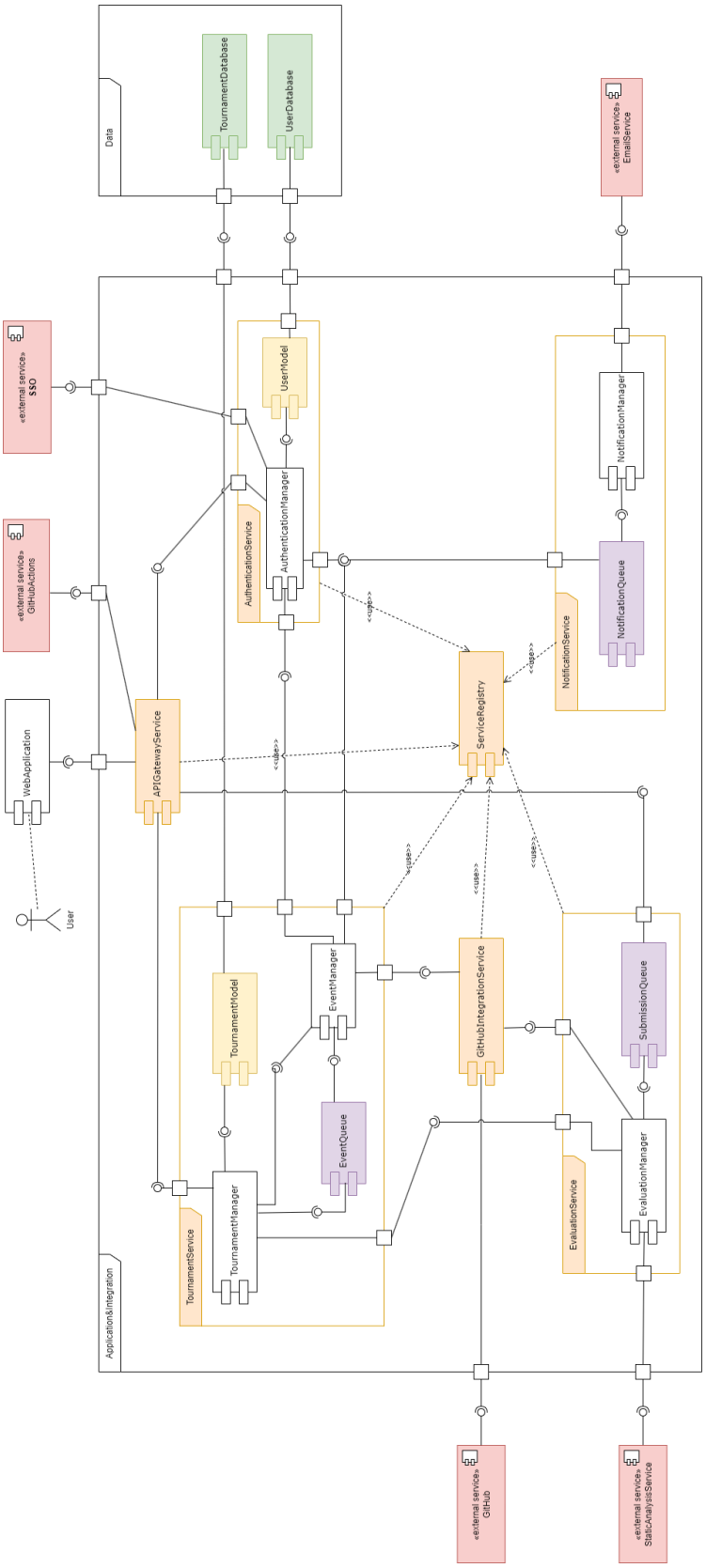

Figure 2: Component diagram

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