POLITECNICO
MILANO 1863

Travlendar+ project YOUR NAMES

Requirement Analysis and Specification Document

Deliverable:	RASD
Title:	Requirement Analysis and Verification Document
Authors:	YOUR NAMES
Version:	1.0
Date:	31-January-2016
Download page:	LINK TO YOUR REPOSITORY
Copyright:	Copyright © 2017, YOUR NAMES – All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, Acronyms, Abbreviations

1.4 Revision History

1.5 Reference Documents

1.6 Document Structure

2 Architectural Design

2.1 Overview

The high-level architectural diagram provided below offers a conceptual overview of the CodeKataBattle (CKB) platform's infrastructure. It delineates the system's division into three primary layers: Presentation, Application, and Data.

The Presentation Layer captures the user interaction with the system via a standard web browser, illustrating the entry point for both educators and students.

The Application Layer is the system's backbone, housing the business logic and core functionalities, including load balancing, application servers, and interfaces for external services such as the GitHub API, Static Analysis Tool API, Email Service, and Notification Service. A dedicated firewall protects this layer, ensuring secure data transactions. It is mainly responsible for handling requests from clients and presentation layer. This layer communicates with the Data Layer, to store and process the data.

The Data Layer is structured to manage persistent data and comprises the Database Management System (DBMS), which supports sharded databases for scalability, and a File Storage system that accommodates various data types, including educator uploads and code submissions. This layer is mainly responsible for data storage and access by querying.

Each component is strategically placed to optimize performance and maintainability, reinforcing the platform's robustness and reliability. The details are discussed in the following sections.

Here, it is important to explain the reasons that led to choice of 3-Tier Architecture. Firstly, this kind of separation of logic helps to improve horizontal scalability. Each layer can be developed and maintained by different software teams. Also, different technologies can be adopted for presentation, application and data layers without affecting each other.

On the other hand, another option can be Microservice Architecture, which is more modular then the 3-layered architecture.It provides higher degree of separation between each part of your application, which leads to even more flexibility and agility than you'd get from a three-tier app. However, as a trade-off, a Microservice Architecture includes more components to deploy and track, which makes developing and maintaining application more challenging because of the higher complexity. In this scenario, even orchestrators and service meshes can be needed. When we think about the CodeKataBattle platform specifically, the scalability of 3-layered architecture is well enough with several servers. A Microservice Architecture might not make sense when a large cluster which maximizes scalability and resilience is not used. Eventhough Microservice Architecture would be better option to scale up and down in a granular way, because of such complexity , it will be excessive for the CodeKataBattle.

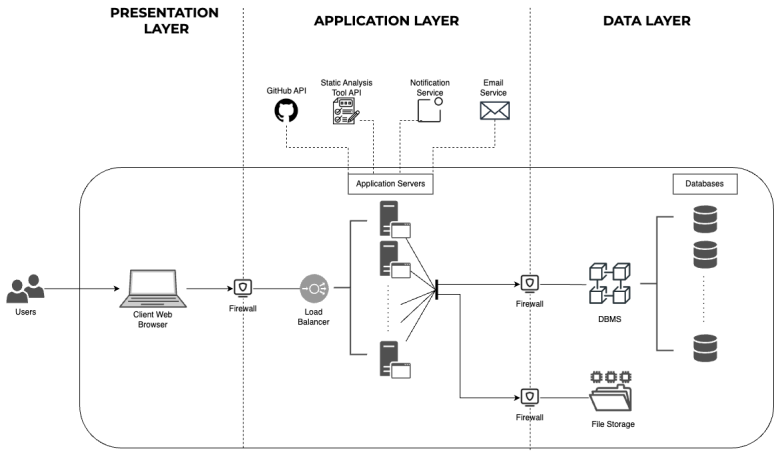

Figure 1: High-Level Architecture of the System

2.2 Component View

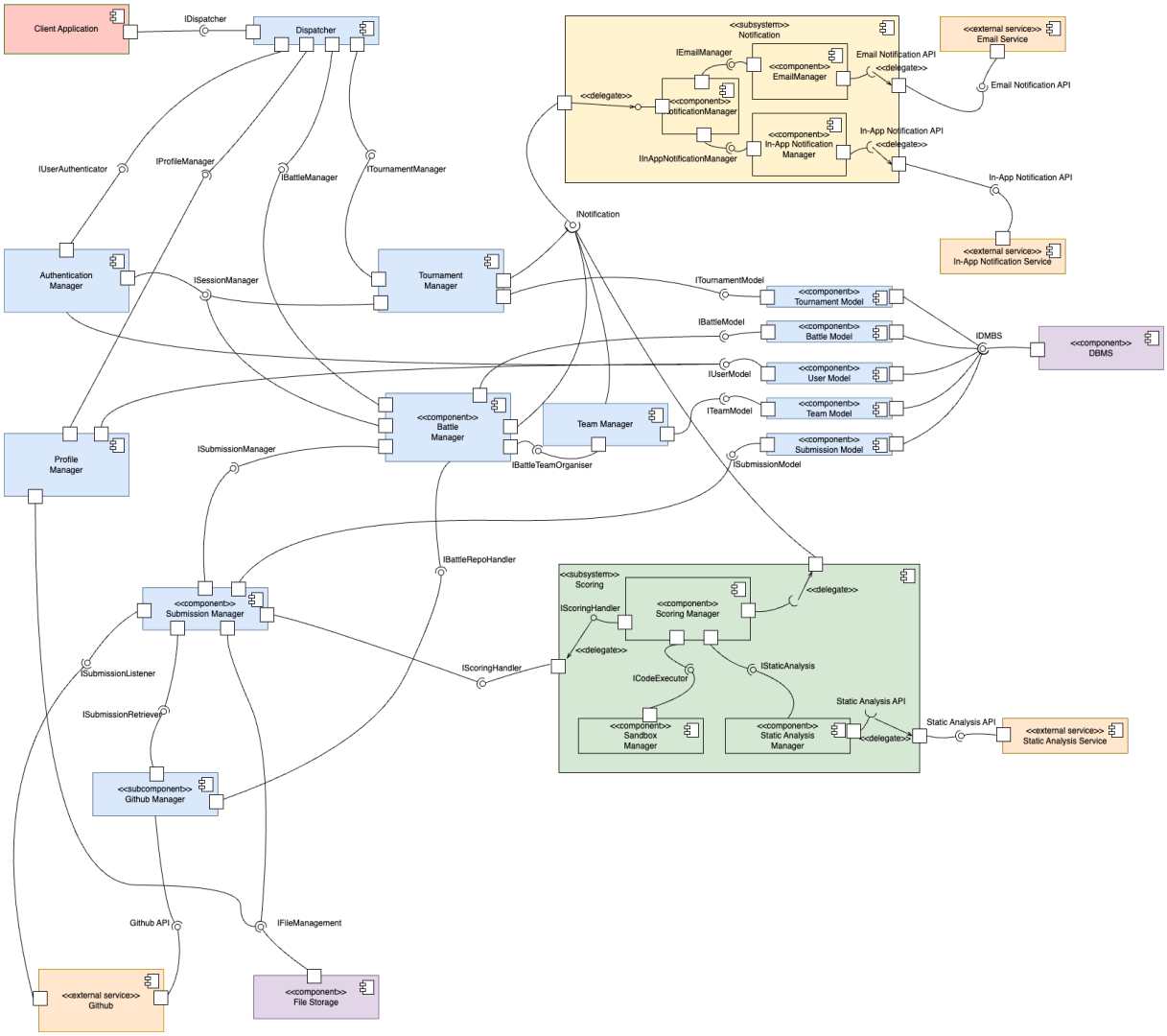

Figure 2: Component Diagram

2.3 Deployment View

Our architecture mainly consists of 3 parts:

- A Static Web Server
- Application Server
- Database Server

,
Users can interact with website via a browser and a device that has browser support such as a computer, a mobile phone etc. Content Delivery Network is used for the web server which behaves as an entry point to users. It hosts the static and dynamic web content, such as .html, .css, .js, and image files, to users. Using Content Delivery Network has some advantages in terms of performance, reliability and security. CDNs speed up content delivery by decreasing the distance between where content is stored and where it needs to go, reducing file sizes to increase load speed, optimizing server infrastructure to respond to user requests more quickly. Also if a server, a data center, or an entire region of data centers goes down, CDNs can still deliver content from other servers in the network. Moreover, it is also very useful from the security perspective. With their many servers, CDNs are better able to absorb large amounts of traffic, even unnatural traffic spikes from a DDoS attack, than a single origin server.

Then we have our Application Server which is hosted on the cloud with 2-4 instances.

At the Data Layer, we have database server which includes database and DBMS with a firewall.

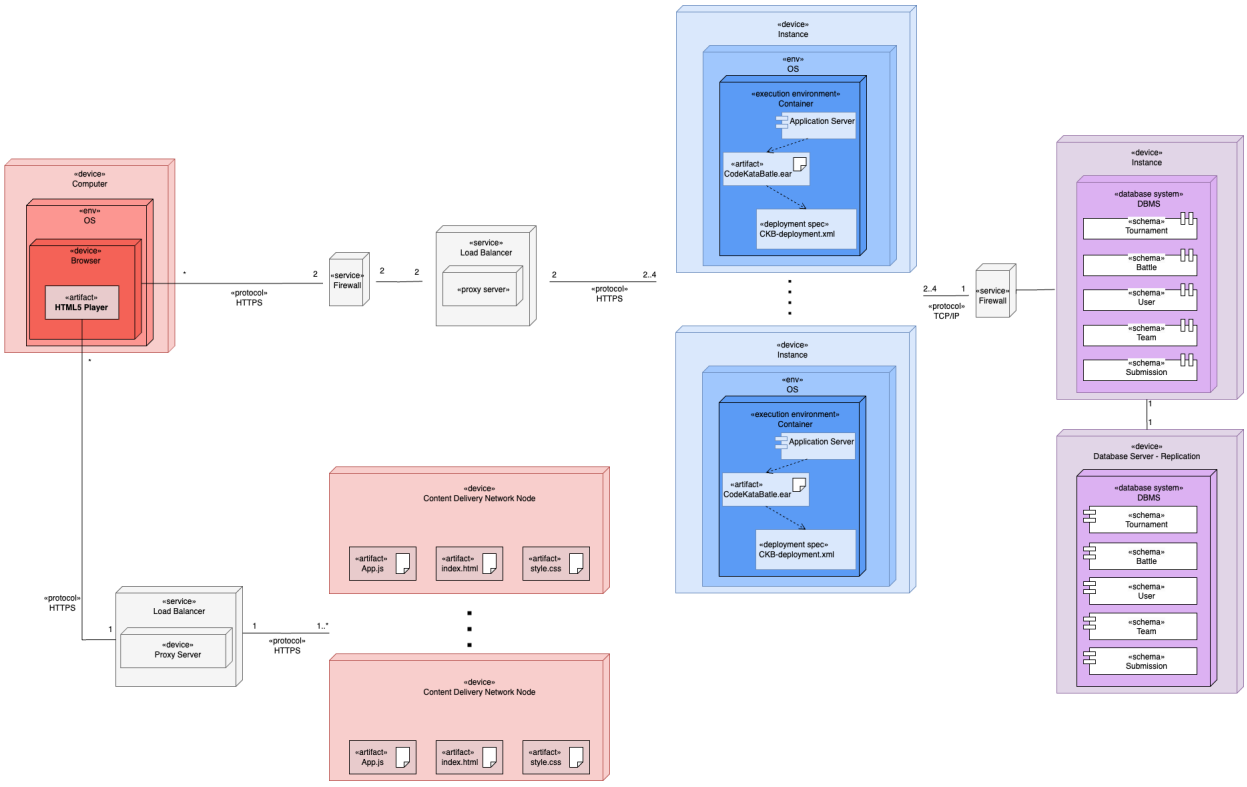

Figure 3: Deployment Diagram

The deployment diagram offers a more detailed view over the hardware and software resources of the application:

User Device: User device can be any device that supports a web browser.

Static Web Content: The static web content of CodeKata Battle is hosted by Content Delivery Network with a Load Balancer distributing traffic and workload. The nodes in CDN can be scaled to very

large numbers because it needs low computation power and memory. Geographically distributed nodes can be helpful to provide web interface with great performance in terms of speed. Briefly, The this content is static and all of its code is run on the client's machine,by browser, so there is no need for any logic to be implemented on the CDN side.

Application Server: All the business logic is handled in the Application Server which is hosted on the cloud with minimum 2 and maximum 4 instances. Obviously, these numbers can change in time but ,for an initial design, using 2-4 server instances would be adequate. Also 2 load balancer is used to distributing workload among application server instances. This is the bottleneck of the application so we decided to use 2 of them.

Distributing incoming requests among multiple servers hosted on the cloud can helps us to fulfill some technical constraints:

Reliability: Cloud providers often offer high reliability through redundant resources and infrastructure. If one server fails, others can take over, minimizing downtime. Regular backups and disaster recovery options further enhance reliability.

Availability: High availability is a key feature of these kind of hosting. Multiple instances in the cloud use multiple data centers around the world, ensuring that your services remain accessible even during local outages or disruptions.

Security: Cloud providers invest heavily in security measures, including physical security of data centers, network security, and data encryption. Monitoring tools and firewall helps to sustain a secure application server.

Scalability: Increasing or decreasing the size and amount of the instances can help to deal with changing request loads that servers have to respond to, also with the help of load balancing. Because of the changes in traffic or workload, we would need different computation and memory capacity. You can easily scale your server resources up or down based on demand, ensuring optimal performance without overpaying for unused capacity.

Maintainability: Cloud providers handle hardware maintenance, updates, and patches, allowing us to focus on core business and application development. Also logging and monitoring tools generally works well with this kind of hosting other than in-house hosting or custom solutions we can develop.

Portability: Cloud environments support portability and interoperability. You can move applications and data across different cloud environments or providers with relative ease, avoiding vendor lock-in and allowing for flexibility in deployment choices.

Database: This instance contains database with necessary schemas and database managements system. It is used with a replication It is also has a replicated version of itself, because of our **reliability** and **availability** constraints. By replicating data across multiple nodes or locations, the system can ensure high **availability**. If one node fails, the others can continue to operate, minimizing downtime and ensuring continuous access to data. Also, in the event of a major failure or disaster affecting the primary data center, having replicas ensures that data is not lost and can be quickly recovered.

Firewalls: Firewall services act as a security gatekeeper for a system's business and data layers, screening incoming connections. They enhance security by enforcing rules that either permit or block traffic, safeguarding the system from illicit access or harmful attacks.

Load Balancers: A load balancer is employed to evenly distribute incoming traffic across various instances of an application. This strategy optimizes the use of resources, boosts performance, and maintains high availability. By doing so, the load balancer aids in managing a significant influx of requests, preventing the application from being overwhelmed or suffering downtime, and contributes to overall stability.

3 User Interface Design

4 Requirements Traceability

5 Implementation, Integration, and Test Plan

6 Effort Spent

References

