

Draw It or Lost It

**CS 230 Project Software Design Template**

Version 1.0

**Table of Contents**

**Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | 9/24/2023 | Chris wactor | Changes made |

**Executive Summary**

The team at The Gaming Room has approached us with a request to create a web-based adaptation of their game, "Draw It or Lose It." Our task will involve crafting a game that possesses features like; the ability to accommodate multiple teams with multiple players within each team, a mechanism to ensure that team names are unique to avoid duplication, and a constraint that ensures only one instance of the game can be active in memory at any given moment.

**Requirements**

1. A game will have the ability to have one or more teams involved.
2. Each team will have multiple players assigned to it.
3. Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
4. Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

**Design Constraints**

The Gaming Room currently offers Draw It or Lose It as an Android app. They've tasked our team at CTS with developing a web-based version of the game. To achieve this, we've decided to utilize Java as our primary technology choice. This selection is based on the fact that Java is already well-suited for Android development, which should streamline the process of creating the web version. We will assess whether any of the tools and components we've used in the Android version can be adapted or extended to function effectively on mobile devices. This approach aims to ensure a seamless gaming experience on both Android devices and the web platform.

**System Architecture View**

Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

**Domain Model**

The Entity superclass serves as the foundation for the Game, Team, and Player classes, enabling them to share common attributes and functions without repetitive coding. These classes, Game Service, Game, Team, and Player, are closely interconnected, forming a flexible association. This association follows a zero too many relationships, allowing them to utilize any number of instances from the others, or none. The Program Driver class takes the lead in managing this package, harnessing the capabilities of the Singleton Tester class as it does so.



**Evaluation**

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client’s requirements and look at the situation holistically, as it all must work together.

In each cell, remove the bracketed prompt and write your own paragraph response covering the indicated information.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| **Server Side** | <Evaluate Mac for its characteristics, advantages, and weaknesses for hosting a web-based software application.> | <Evaluate Linux for its characteristics, advantages, and weaknesses for hosting a web-based software application.> | <Evaluate Windows for its characteristics, advantages, and weaknesses for hosting a web-based software application.> | <Evaluate Mobile Devices for their characteristics, advantages, and weaknesses for hosting a web-based software application.> |
| **Client Side** | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Mac.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Linux.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Windows.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Mobile Devices.> |
| **Development Tools** | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Mac.> | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Linux.> | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Windows.> | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Mobile Devices.> |

**Recommendations**

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

* **Operating Platform**: I would recommend Kubernetes for the operating platform. Given that “Draw It or Lose It” will be looking to expand into other environments, it would be best to run with a platform like Kubernetes for its containerized workload skills. While other platforms could do the same, Kubernetes is used to manage multiple containers.
* **Operating Systems Architectures**: Well, the first reason would be scalability. Kubernetes is capable of adjusting the container count dynamically regulated by the workload. This guarantees that "Draw It or Lose It" can accommodate fluctuating player numbers without needing manual adjustments. Multi-cloud flexibility would be another reason Kubernetes Is the operating system for this game. Given that some Cloud providers like Google Cloud or azure Might have better infrastructure in different regions of the world, multi-cloud flexibility is needed so “Draw It or Lose It” isn’t tied to a single cloud provider.
* **Storage Management**: Since Kubernetes Is the recommended operating platform, I would say use storage classes within Kubernetes. The benefit is having dynamic provisions when needed along with versatility, Cloud integration and cost efficiency. Kubernetes storage classes were always designed with cloud environments in mind, so seamlessly integrating with cloud provider storage solutions and keeping cost down by prioritizing different storage classes depending on the needs to optimize without compromising performance.
* **Memory Management**: Kubernetes employs a multifaceted approach to memory management. Defining resources like “requests” and “limits” to ensure consistent application performance. Being able to set the minimum resource required by predictable resource allocation Helps to ensure a container gets only what’s required. You can even set limits on the memory and prevent a container from consuming all available memory, inevitably causing a leak.
* **Distributed Systems and Networks**: While distributed systems bring scalability and flexibility, they will also introduce complexities to the system. Load balancing and connectivity to name a few. A load balancer for server traffic is important by ensuring no server gets overwhelmed. You would have to balance and distribute all traffic across multiple servers. The challenge with connectivity is some platforms may deal with weaker connections, causing stability issues in the program.
* **Security**: Kubernetes has many built-in safety features, but it's important to add more layers of protection at the app and internet connection levels. Encrypt users' data stored in databases by using data at rest encryption. Encrypt data as it moves between client devices and the game services by using data in transit encryption. So, protect the game’s API from malicious attacks by using an API security. Offer a rate limit by implementing the API Gateway, then Work on authorization and request validation. By looking at everything - from coding safe locks to deciding who gets access, to keeping an eye on the system - "Draw It or Lose It" can keep gamers safe no matter where they play.