

Draw It or Lose It

**CS 230 Project Software Design Template**

Version 2.0

**Table of Contents**

**Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 3.0 | 06/22/2024 | Besker Telisma | Updated table, recommendations for project Three |

**Instructions**

Fill in all bracketed information on page one (the cover page), in the Document Revision History table, and below each header. Under each header, remove the bracketed prompt and write your own paragraph response covering the indicated information.

**Executive Summary**

Creative Technology Solutions (CTS) is evaluating the deployment of the game application, "Draw It or Lose It," on multiple platforms for The Gaming Room. This document evaluates Linux, Mac, Windows, and mobile platforms for their characteristics, advantages, and weaknesses in hosting a web-based application and supporting various clients. The goal is to provide a comprehensive understanding to help The Gaming Room make an informed decision on expanding their game to multiple platforms.

**Requirements**

**Business Requirements**

* Multiple Teams: The game must support the ability to have one or more teams involved in a game session.
* Multiple Players per Team: Each team must be able to have multiple players assigned to it.
* Unique Game and Team Names: To avoid confusion and ensure a smooth user experience, game and team names must be unique. Users must be able to check whether a name is already in use when choosing a game or team name.
* Single Instance in Memory: Only one instance of the game can exist in memory at any given time. This ensures consistency and prevents conflicts in game state management.

**Technical Requirements**

* Unique Identifiers: Each instance of a game, team, or player must have unique identifiers to maintain uniqueness and integrity of the data.
* Singleton Pattern: Implement the Singleton pattern for the GameService class to ensure that only one instance of the class exists in memory at any time.
* Iterator Pattern: Use the Iterator pattern to complete the addGame(), getGame(), addTeam(), and addPlayer() methods. This ensures efficient and reliable traversal of collections to check for uniqueness and manage game entities.
* Base Entity Class: Create a base class called Entity to hold common attributes (id and name) and behaviors for games, teams, and players.
* Inherit from Entity: Refactor the Game, Team, and Player classes to inherit from the Entity class, ensuring consistency and code reuse.
* Web-Based Distributed Environment: The game must be developed as a web-based application, capable of serving multiple platforms including various web browsers and devices.

**Design Constraints**

* Scalability: The application must support multiple simultaneous users and games.
* Performance: Real-time drawing updates and guessing must be responsive and smooth.
* Security: User data must be protected, and only authorized users should access specific game features.
* Compatibility: The application must work across different web browsers and devices.
* Maintainability: The codebase should be easy to maintain and extend, following best practices and design patterns.

**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 UML class diagram represents the domain model for the game application. Key classes include:

* Entity Class: A base class with common attributes id and name. It ensures that all derived classes (Game, Team, Player) have a unique identifier and a name.
* GameService Class: Implements the Singleton pattern to ensure only one instance exists in memory. Manages the creation and retrieval of Game, Team, and Player instances using unique identifiers.
* Game Class: Inherits from Entity and manages a list of Team instances. Ensures team names are unique within a game.
* Team Class: Inherits from Entity and manages a list of Player instances. Ensures player names are unique within a team.
* Player Class: Inherits from Entity and represents individual players.
* ProgramDriver Class: Used for running and testing the application.

Object-oriented programming principles used:

* Inheritance: Entity class provides common attributes and methods for Game, Team, and Player classes.
* Encapsulation: Each class manages its state and behavior, exposing only necessary methods.
* Singleton Pattern: Ensures a single instance of GameService to manage the application's state.
* Iterator Pattern: Used in GameService to iterate through games, teams, and players efficiently.



**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 has to 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** | Mac offers good development tools such as Xcode, which provides a robust environment for developing server applications with a Unix-based stability. However, it can be costly for server deployment. | Linux is cost-effective, highly customizable, and widely used for server deployments. It offers excellent performance and stability, especially for large-scale applications. | Windows Server provides enterprise-level features and extensive support. However, it may require more resources and higher licensing costs. | Mobile devices are not typically used for server hosting due to limited resources and the need for constant uptime and high availability. |
| **Client Side** | Mac provides a seamless development experience for iOS applications with tools like Xcode, ensuring compatibility with Safari and other Apple browsers | Linux offers various open-source tools for web development and is highly compatible with browsers like Firefox and Chrome. | Windows supports a wide range of development tools such as Visual Studio and is essential for developing Windows applications. It is compatible with Internet Explorer and Edge. | Mobile devices require responsive web design and possibly native app development using tools like Android Studio for Android and Xcode for iOS. |
| **Development Tools** | Xcode, Swift, Objective-C for Mac and iOS development. | Eclipse, IntelliJ, VS Code, Java, Python for web and server-side development. | Visual Studio, .NET, Java, various IDEs for Windows development | Android Studio, Xcode, React Native, Flutter for mobile app development. |
| **Server-Based Deployment** | Yes, macOS supports server deployment but is less common. | Yes, widely supported for server deployments. | Yes, widely supported for server deployments. | Not applicable for server deployment. |
| **Licensing Costs** | High cost for macOS server licenses.  ***Tools Cost:***   High cost for development tools and Apple hardware. | Generally free, but enterprise versions (e.g., Red Hat) have costs.  ***Tools Cost:***  Mostly free and open-source, potential costs for premium IDE features. | High licensing costs for Windows Server.  ***Tools Cost:***  High licensing costs for some tools (e.g., Visual Studio Enterprise). | Not applicable.  ***Tools Cost:***  Free IDEs available, but potential costs for premium tools or services. |
| **Development Considerations** | Higher development costs due to expensive hardware and tools. | Cost-effective with open-source tools; may require expertise in Linux. | Extensive support and tools; potential need for multiple versions to ensure compatibility. | Requires knowledge of mobile frameworks and ensuring compatibility across multiple devices. |
| **Impact on Development Team** | Requires Mac expertise and access to Mac hardware. | Requires familiarity with Linux; generally lower costs for development tools. | Requires knowledge of Windows environments; potential for higher tool costs. | Requires expertise in mobile development for iOS and Android. |
|  |  |  |  |  |

**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**: Linux is my recommended for server-side deployment due to its cost-effectiveness, stability, and customization options. For client-side development, both Mac and Windows are suitable depending on the target audience (iOS for Mac, Windows applications for Windows).
* **Operating Systems Architectures**:

1. Linux architecture supports high concurrency and scalability, making it ideal for web servers. It is highly customizable and provides excellent performance and stability.
2. Mac provides a robust environment for client-side development, particularly for iOS applications. It offers a seamless development experience with tools like Xcode.
3. Windows provides a comprehensive environment for client-side development, particularly for Windows applications. It offers extensive support and development tools.

* **Storage Management**: Use a relational database (e.g., MySQL, PostgreSQL) for structured data management and NoSQL databases (e.g., MongoDB) for unstructured data, also it's important to ensure regular backups and disaster recovery plans are in place.
* **Memory Management**: Implement efficient memory management techniques such as caching frequently accessed data, using memory pools, and optimizing queries to reduce memory usage.
* **Distributed Systems and Networks**:

1. Use RESTful APIs for communication between different platforms.
2. Implement load balancers and failover mechanisms to ensure high availability and fault tolerance.
3. Ensure robust network connectivity to handle the dependencies between the components within the distributed systems.

* **Security**: Implement HTTPS for secure communication, use authentication and authorization mechanisms (e.g., OAuth2), and encrypt sensitive data at rest and in transit. Regularly update and patch software to address security vulnerabilities.