

Draw It or Lose It

**CS 230 Project Software Design Template**

Version 1.0

**Table of Contents**

**Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | 08/16/24 | Nicholas Dalton | Initial creation of the software design template. |

**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**

The Gaming Room is expanding its current Android-based game, Draw It or Lose It, into a web-based application that serves multiple platforms. The game, similar to the 1980s TV show Win, Lose or Draw, involves teams guessing a puzzle based on stock drawings. This expansion requires a comprehensive software design to ensure the game can be played seamlessly across different platforms while maintaining the unique team and game instance constraints. This document outlines the software design and development approach, ensuring the game meets all specified requirements and performs efficiently in a web-based environment.

**Requirements**

Business Requirements:

Seamless expansion from Android to web-based platforms.

Ability for multiple teams and players to interact simultaneously.

Ensuring unique game, team, and player names.

Technical Requirements:

Implementation of real-time updates for all connected clients.

Cross-platform compatibility, including Windows, Mac, Linux, and mobile devices.

Adherence to single-instance constraints for game instances.

**Design Constraints**

Developing the game application in a web-based distributed environment presents several design constraints:

Scalability: The application must handle multiple teams and players without performance degradation.

Uniqueness: Game, team, and player names must be unique to prevent conflicts.

Single Instance: Only one instance of the game should exist in memory at any given time.

Real-time Updates: The game must update in real-time to reflect changes across all connected clients.

Cross-Platform Compatibility: The application must be compatible with various operating systems and devices, including Windows, Mac, Linux, and mobile platforms.

These constraints impact the development process by requiring robust backend architecture, efficient data handling, and ensuring compatibility across different platforms and devices.

**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 provided shows the relationships between various entities in the game application. Key classes include:

Entity: A base class holding common attributes (id, name) and behaviors.

Game: Inherits from Entity, represents a game instance.

Team: Inherits from Entity, represents a team participating in the game.

Player: Inherits from Entity, represents an individual player in a team.

Object-oriented programming principles are demonstrated through inheritance (e.g., Game, Team, and Player classes inheriting from Entity), encapsulation (each class contains its own attributes and methods), and the singleton pattern (ensuring only one instance of the game exists in memory).



**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 stable performance but may require additional configuration for scalability. | Linux is highly stable, scalable, and preferred for server-side applications. | Windows provides good performance but can be costlier due to licensing fees. | Mobile devices are not ideal for server-side but necessary for client-side support. |
| **Client Side** | Requires consideration of compatibility and user experience on Mac devices. | Linux clients are less common; focus on cross-browser compatibility. | Ensure application works smoothly on various Windows versions. | Essential for mobile user experience; ensure app is responsive and user-friendly. |
| **Development Tools** | Xcode, Java, and relevant IDEs for Mac development. | Eclipse, NetBeans, Java, and other tools for Linux development. | Visual Studio, Java, and other tools for Windows development. | Android Studio, Xcode (for iOS), and cross-platform tools like React Native. |

**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:**

Recommendation:

For the server, I recommend using Linux because it's known to be reliable, can easily handle a lot of users, and is free, which helps keep costs down. On the user side, using web technologies like HTML, CSS, and JavaScript will allow the game to work on different devices and operating systems, like Windows, macOS, and smartphones.

**Operating Systems Architectures:**

Details:

Linux will be used on the server to manage the backend of the game, while the frontend (what the users see) will be available on all major operating systems through web browsers. This way, the game can be played on Windows, macOS, Linux, and even mobile devices.

**Storage Management:**

System:

For storing game data, a relational database like MySQL or PostgreSQL would be a good choice. These databases are good at keeping data organized and making sure it can be accessed quickly and reliably.

**Memory Management:**

Techniques:

On the Linux server, it's important to manage memory efficiently, so using caching (storing frequently accessed data in a special memory) can help. This makes the game run faster because the server doesn't have to constantly go back to the main database for every piece of information.

**Distributed Systems and Networks:**

Communication:

To make sure the game works well on different devices, we can use RESTful APIs, which allow the server and clients (the devices running the game) to communicate easily. For real-time updates, WebSockets can be used so that all players see the game updates as they happen, without delays.

**Security:**

User Protection:

Security is really important, so the server will use HTTPS to make sure all communication between the server and clients is encrypted and safe. We'll also use authentication methods like OAuth to make sure only authorized users can access the game. Additionally, we'll follow best practices for encrypting data and writing secure code to protect user information across different platforms.