

<Name-of-Software-Application>

**CS 230 Project Software Design Template**

Version 1.0

**Table of Contents Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | 3/20/2025 | Khris Tarin | Initial draft of the software design document. |

**Executive Summary**

The Gaming Room seeks to expand its existing mobile game, Draw It or Lose It, into a web-based

application that can be accessed across multiple platforms. This transition presents a unique software design challenge, as the application must support multiple teams and players, ensure unique naming conventions for games and teams, and maintain a single instance of the game in memory at any time. To address these requirements, we propose a robust architecture that leverages modern web technologies, ensuring scalability, maintainability, and a seamless user experience. Our solution will include a well- defined structure for managing game instances, teams, and players, while also implementing necessary security measures to protect user data. This document outlines the design constraints, domain model, and evaluation of potential operating platforms to guide the development process.

**Requirements**

The client has outlined several business and technical requirements for the web-based game application:

* The game must support one or more teams, each with multiple players.
* Game and team names must be unique to prevent conflicts.
* Only one instance of the game can exist in memory at any time, necessitating unique identifiers for games, teams, and players.
* The application should be accessible across various platforms, including web browsers and mobile devices.

**Design Constraints**

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

* Scalability: The application must handle varying numbers of users and teams without

performance degradation. This requires careful planning of server resources and load balancing.

* Cross-Platform Compatibility: The game must function seamlessly across different operating systems and devices, necessitating the use of responsive design and web standards.
* Real-Time Interaction: The game requires real-time updates for player interactions, which imposes constraints on the choice of technologies (e.g., WebSockets for real-time communication).
* Security: Protecting user data and ensuring secure communication between clients and servers is paramount, requiring the implementation of encryption and secure authentication methods.
* Single Instance Management: The application must ensure that only one instance of the game is active at any time, which complicates state management and requires a robust backend

architecture.

**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 for the game application consists of several key classes that represent the core components of the game. The primary classes include:

* Game: Represents the game instance, containing attributes such as game ID, game name, and a list of teams. It manages the game state and ensures that only one instance exists in memory.
* Team: Represents a team participating in the game, with attributes for team ID, team name, and a list of players. The uniqueness of team names is enforced at this level.
* Player: Represents individual players within a team, with attributes for player ID, player name, and team affiliation. Players are associated with a specific team, ensuring that team compositions are maintained.
* Entity: A base class that holds common attributes and behaviors shared among Game, Team, and Player classes, promoting code reusability and adherence to the DRY (Don't Repeat Yourself)

principle.

* The relationships between these classes demonstrate key object-oriented programming

principles, such as encapsulation (by keeping attributes private and providing public methods for access) and inheritance (with the Entity class serving as a base for other classes). This design

efficiently fulfills the software requirements by ensuring clear organization and management of game components.



**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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Development**  **Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| **Server Side** | Mac offers a stable environment for hosting web  applications, but its cost can be a disadvantage for large-scale  deployments. It is less commonly used for server- side applications compared to Linux. | Linux is widely regarded as the best choice for hosting web applications due to its stability,  security, and cost- effectiveness. It supports a variety of web servers and programming languages. | Windows  provides a user- friendly  environment and is compatible  with many  development tools, but it can be more expensive and less efficient for server-side  applications compared to  Linux. | Mobile devices can host web  applications  through browsers, but performance may vary based on device  specifications. They are not ideal for hosting but are essential for client- side access. |
| **Client Side** | Developing for Mac requires  expertise in | Linux  development can  be cost-effective, | Windows is widely used,  making it easier | Mobile  development  requires knowledge |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | macOS-specific tools and  frameworks, which may increase costs and development time. | but it may require specialized knowledge of Linux-based tools and  environments. | to find developers with the  necessary expertise.  However, it may incur higher  licensing costs. | of responsive design and cross- platform  frameworks, which can increase  development time  and complexity |
| **Development Tools** | Common tools include Xcode for macOS  development, along with languages like Swift and Objective-C | Popular tools include various IDEs like Visual Studio Code, and languages such as Python, Ruby, and PHP | Development tools include Visual Studio and .NET  frameworks, with support for  languages like C# and JavaScript. | Mobile  development tools include React Native, Flutter, and various web  technologies (HTML, CSS,  JavaScript) for cross-platform  compatibility. |

**Recommendations**

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

* **Operating Platform:** Linux accommodates a variety of architectures, such as x86 and ARM, providing deployment flexibility. Its modular architecture facilitates straightforward updates and maintenance, allowing the application to scale in response to increasing user demand.
* **Operating Systems Architectures**: Linux supports multiple architectures, including x86 and ARM, which enhances deployment flexibility. Its modular structure allows for easy updates and maintenance, ensuring that the application can grow alongside user demand.
* **Storage Management**: For effective storage management, I recommend utilizing a relational database management system (RDBMS) like PostgreSQL or MySQL. These systems offer strong data integrity, support for complex queries, and scalability, which are crucial for managing game data, user accounts, and game states..

**Memory Management**: Linux utilizes advanced memory management techniques, such as virtual memory and paging, to optimize the performance of the Draw It or Lose It software. This ensures that the application can accommodate multiple concurrent users without excessive resource usage.

**Distributed Systems and Networks**To enable communication across different platforms, the application can leverage RESTful APIs or WebSockets for real-time interactions. This method allows for smooth data exchange between clients and servers, ensuring synchronized game states across devices. The architecture should also consider potential network disruptions by incorporating retry mechanisms and fallback strategies.

* **Security:** To safeguard user information, the application should implement HTTPS for secure communication, along with strong authentication methods (e.g., OAuth2). Furthermore, user data must be encrypted both during transmission and while stored, ensuring compliance with data protection regulations and maintaining user privacy across platforms.