

Draw It Or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/21/2024 | August Moews | First version of software design for Draw It Or Lose It |

**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](#_sbfa50wo7nsh)

The Gaming Room is a successful web-based game application that is seeking to expand its reach by developing a mobile application for both Android and iOS platforms. The client has outlined four primary objectives for this project:

1. Develop a mobile version of the existing game to increase user accessibility and engagement.
2. Maintain compatibility with both Android and iOS devices, ensuring broad user adoption.
3. Address any platform-specific constraints related to mobile environments, including memory, storage, and security.
4. Transition seamlessly from the existing web platform to a mobile application while preserving the core functionalities and user experience of the original system.

This project is constrained by the client’s budget, platform-specific challenges, and the increased complexity of designing and deploying applications for mobile environments compared to the web platform. The design process will focus on delivering a robust, scalable, and secure mobile application while adhering to the outlined requirements and constraints.

## Requirements

The requirements for The Gaming Room project are explicitly stated in the assignment prompt. These requirements outline the expected functionalities and outcomes for the mobile application. They are as follows:

1. **Develop Mobile Compatibility**: The application must be designed for both Android and iOS platforms to ensure accessibility for a broader user base.
2. **Maintain Core Functionality**: The mobile application must replicate the existing features of the web-based platform, preserving the gameplay experience and user interface.
3. **Support Cross-Platform Accessibility**: The solution must seamlessly integrate the current web-based environment with the mobile platforms, allowing users to switch between devices without data loss or functionality issues.
4. **Adhere to Budgetary Constraints**: The development and deployment of the mobile application must remain within the client’s predefined budget, ensuring cost-effective solutions without compromising quality.

## [Design Constraints](#_2et92p0)

The Gaming Room project presents several design constraints that must be addressed to meet the client’s requirements effectively. These constraints focus on the challenges inherent in transitioning from a web-based platform to a mobile application and operating within the specified parameters of mobile environments.

1. **Development Lifecycle Environments**:
   1. The project will require multiple environments to support the development lifecycle, including research, development, testing (QA and user acceptance testing), and staging environments. Establishing and maintaining these environments may introduce logistical and resource challenges.
2. **Security Considerations**:
   1. The shift to mobile platforms introduces new security challenges. Ensuring the protection of user data, secure communication channels, and adherence to platform-specific security requirements is essential.
3. **Memory and Storage Limitations**:
   1. Mobile devices, particularly older models, may have limited memory and storage capabilities compared to the web platform. This constraint necessitates careful consideration of application size and resource utilization.
4. **Increased Application Complexity**:
   1. Transitioning from a web-based application to a mobile application increases development complexity. The need to support multiple operating systems (iOS and Android) and their associated frameworks and hardware configurations adds to the complexity of design and implementation.
5. **Budget Constraints**:
   1. The predefined budget limits the resources available for development, testing, and deployment. This constraint requires prioritization of features and infrastructure needs while ensuring quality and performance standards.

## [System Architecture View](#_ilbxbyevv6b6)

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](#_8h2ehzxfam4o)

The Domain Model represents the structure of the game application using a UML class diagram. This diagram illustrates the key entities and their relationships within the system.

1. Class Overview:

* Entity Clas: A base class designed to hold common attributes (id and name) and behaviors for all entities in the system. This class centralizes shared functionality and enables efficient reuse.
* Game Class: Represents an individual game. It inherits from Entity and includes functionality specific to managing a game instance.
* Team Class: Represents a team participating in a game. It inherits from Entity and contains attributes and methods to manage teams and players.
* Player Class: Represents a player on a team. It inherits from Entity and includes player-specific functionality.
* GameService Class: Acts as the central manager for games, teams, and players. This class implements the Singleton and Iterator design patterns to manage instances and ensure scalability and resource efficiency.

1. Relationships Between Classes:

* Inheritance: The Entity class serves as the parent class for Game, Team, and Player. This reduces redundancy by centralizing shared attributes and behaviors.
* Association:
* A Game contains multiple Team objects, represented as a 1..\* relationship.
* A Team contains multiple Player objects, also represented as a 1..\* relationship.
* Aggregation: The GameService class aggregates collections of Game, Team, and Player instances, allowing centralized management.

1. Object-Oriented Programming Principles:

* Encapsulation:
* Each class hides its internal state using private fields and provides controlled access via public getter methods.
* Inheritance:
* The Entity class allows common attributes (id, name) to be shared among subclasses, reducing redundancy.
* Polymorphism:
* By inheriting from Entity, subclasses (Game, Team, Player) can override methods (e.g., toString()) to provide class-specific behavior.

1. Deisgn Patters in Use:

* Singleton:
* The GameService class is implemented as a Singleton to ensure only one instance exists in memory, providing centralized control over the application’s logic.
* Iterator:
* The GameService class uses the Iterator pattern to manage collections of games, teams, and players, allowing efficient traversal and enforcement of unique names.

1. Alignment with Requirements:

* The GameService class ensures unique names for games and teams using the Iterator pattern.
* Unique identifiers are automatically assigned using static counters in the GameService class.
* The Entity base class centralizes common attributes, simplifying future extensions.

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## [Evaluation](#_2o15spng8stw)

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 systems are known for their stability and seamless integration with other Apple devices. However, they are less commonly used as servers due to their higher cost and limited server-oriented tools. Mac servers are reliable for small-scale applications but might not scale well for enterprise-level operations. | |  | | --- | |  | | Linux is highly preferred for server environments due to its flexibility, stability, and cost-effectiveness. Its open-source nature allows for customization to meet specific project requirements, such as the development of a game application. However, Linux systems may require advanced technical expertise for setup and maintenance. | | Windows servers provide strong compatibility with enterprise software and tools, making them an excellent choice for organizations familiar with Microsoft environments. Their GUI-driven interface makes them user-friendly for administrators. However, licensing costs and potential resource requirements can be drawbacks for budget-constrained projects. | Mobile devices are not typically used as servers due to their limited processing power, storage, and lack of server-grade operating systems. While they can host lightweight applications or local development environments, they are unsuitable for robust server requirements like handling game data and user management. |
| **Client Side** | Developing for Mac clients requires tools like Xcode, which provides a robust environment for building applications for macOS. However, development can be costly due to the need for Apple hardware and the Apple Developer Program subscription. Time and expertise are also required to understand macOS-specific APIs and guidelines. | Linux offers a cost-effective development environment due to its open-source tools and software. Many distributions provide flexibility for testing across different configurations. However, supporting Linux clients may require specialized expertise due to the variety of distributions and a lack of uniformity across environments. | Windows is a widely adopted client platform, making it critical to support. Development for Windows benefits from tools like Visual Studio, which integrates seamlessly with .NET frameworks. However, licensing fees and system requirements for development tools could increase costs. Expertise in Windows-specific frameworks is also necessary. | Supporting mobile clients requires development for Android and iOS platforms using tools like Android Studio, Kotlin, Swift, or cross-platform frameworks such as Flutter. While mobile development can expand the user base significantly, it involves challenges such as ensuring compatibility across devices, adhering to platform-specific guidelines, and managing longer testing cycles. |
| **Development Tools** | Development on macOS requires tools such as Xcode, which is the official IDE for macOS and iOS development. It supports languages like Swift and Objective-C, providing robust debugging and performance analysis tools. However, it is exclusive to macOS, requiring developers to use Apple hardware. | Linux offers a wide variety of open-source development tools, including Eclipse, IntelliJ IDEA, and NetBeans. It supports programming languages like Java, Python, and C++. Linux-based systems are ideal for command-line tools, scripting, and server-side development. The challenge lies in the fragmentation of Linux distributions, which may necessitate additional testing. | Windows development relies heavily on Visual Studio, a powerful IDE that supports multiple languages such as C#, C++, and Python. The Windows ecosystem also integrates well with .NET for building applications. However, licensing fees for Visual Studio Professional or Enterprise versions can add to development costs. | Mobile development tools include Android Studio for Android apps and Xcode for iOS apps. Cross-platform frameworks like Flutter or React Native can simplify development by allowing code reuse across Android and iOS. However, these frameworks may have limitations in fully leveraging native device capabilities, requiring additional platform-specific work. |

## Recommendations

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

1. **Operating Platform**: Linux
2. **Operating Systems Architectures**: The chosen Linux-based architecture for The Gaming Room leverages a modular kernel design for lightweight, efficient operation and customization (OSDev Wiki, n.d.). Its robust multitasking and process scheduling capabilities ensure smooth concurrent execution of essential services like databases and APIs. Linux supports containerization tools such as Docker and Kubernetes, enabling consistent development and deployment across environments (DataCamp, 2024). For storage, the EXT4 file system provides stability and high performance, while networked file systems like NFS support distributed storage needs (Timescale, n.d.). Additionally, Linux’s advanced networking features ensure secure and reliable communication between distributed components, making it an ideal platform for scalable and secure game application deployment (GeeksforGeeks, 2024).
3. **Storage Management**: The recommended storage management solution for The Gaming Room combines PostgreSQL for managing structured data like user accounts and game statistics, ensuring reliability and scalability (Timescale, n.d.). Cloud-based storage, such as AWS S3, is ideal for storing large assets like graphics and audio files, offering redundancy and high availability (Amazon Web Services, n.d.). Locally, the Linux EXT4 file system ensures fast and stable read/write performance, while networked file systems like NFS can support distributed file sharing between servers (Timescale, n.d.). This hybrid approach provides a robust and scalable storage solution tailored to the project’s needs.
4. **Memory Management**: Linux’s memory management techniques make it an ideal platform for hosting Draw It or Lose It (The Linux Kernel Documentation, n.d.). Features like demand paging ensure only necessary data is loaded into memory, while virtual memory allows processes to use more memory than physically available through swap space. Robust caching speeds up frequently accessed data, and process isolation ensures stability in a multi-user environment (The Linux Kernel Documentation, n.d.). Additionally, support for garbage collection in modern programming languages optimizes resource use, ensuring smooth and efficient performance under high user demand.
5. **Distributed Systems and Networks**: Linux’s robust networking capabilities make it ideal for distributed systems like Draw It or Lose It. With support for TCP/IP, RESTful APIs, and WebSocket protocols, it facilitates real-time communication between platforms. Tools like NGINX and HAProxy enable load balancing, while clustering and failover mechanisms ensure fault tolerance and reliability. SSL/TLS encryption provides secure communication between components (GeeksforGeeks, 2024), and lightweight protocols address platform-specific constraints. Together, these features ensure seamless interaction and performance across web and mobile platforms, even under varying network conditions.
6. **Security**: Security is a critical component for Draw It or Lose It, and Linux provides robust features to protect user information across platforms. Using SSL/TLS encryption, data is secured during transmission, ensuring confidentiality and integrity (GeeksforGeeks, 2024). Linux’s built-in firewall tools, like iptables and nftables, safeguard against unauthorized access and network threats (LinuxConfig.org, n.d.). Secure authentication methods, such as OAuth or token-based systems, can manage user sessions effectively. Additionally, Linux supports role-based access control (RBAC) to restrict system and data access based on user roles, ensuring protection against internal and external breaches (Linux Foundation Documentation, n.d.). These measures collectively provide a strong security foundation for the platform.

REFERENCES:

Amazon Web Services. (n.d.). Amazon S3 - Cloud object storage. *Amazon Web Services*. Retrieved December 2, 2024, from <https://aws.amazon.com/s3/>

DataCamp. (2024, November). Kubernetes vs Docker: Differences every developer should know. *DataCamp*. Retrieved December 2, 2024, from [https://www.datacamp.com/blog/kubernetes-vs-](https://www.datacamp.com/blog/kubernetes-vs-docker) docker

GeeksforGeeks. (2024, July). Secure communication in distributed system. *GeeksforGeeks*. Retrieved December 2, 2024, from [https://www.geeksforgeeks.org/secure-communication-in-distributed-](https://www.geeksforgeeks.org/secure-communication-in-distributed-system/) system/

LinuxConfig.org. (n.d.). Advanced firewall management with nftables: Transitioning from iptables. *LinuxConfig.org*. Retrieved December 2, 2024, from [https://linuxconfig.org/advanced-firewall-](https://linuxconfig.org/advanced-firewall-management-with-nftables-transitioning-from-iptables) management-with-nftables-transitioning-from-iptables

Linux Foundation Documentation. (n.d.). Role-based access control. *Linux Foundation Documentation*. Retrieved December 2, 2024, from [https://docs.linuxfoundation.org/lfx/project-control-](https://docs.linuxfoundation.org/lfx/project-control-center/v1-prior-version/role-based-access-control) center/v1-prior-version/role-based-access-control

OSDev Wiki. (n.d.). Modular kernel. *OSDev Wiki*. Retrieved December 2, 2024, from <https://wiki.osdev.org/Modular_Kernel>

The Linux Kernel Documentation. (n.d.). Memory management. *The Linux Kernel Documentation*. Retrieved December 2, 2024, from [https://www.kernel.org/doc/html/latest/admin-](https://www.kernel.org/doc/html/latest/admin-guide/mm/index.html) guide/mm/index.html

Timescale. (n.d.). Structured vs. semi-structured vs. unstructured data in PostgreSQL. *Timescale*. Retrieved December 2, 2024, from <https://www.timescale.com/learn/structured-vs-semi-structured-vs-unstructured-data-in-postgresql>