

Draw It or Lose It

# **CS 230 Project 2 Software Design Template**

Version 2.0

## 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 |
| --- | --- | --- | --- |
| 2.0 | 11/26/23 | Michael Walston | This is the second version of this game, focus more on the development API and software. |

**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 wants to develop a web-based game that can run on multiple platforms. The game will be called Draw It or Lose It and is currently only available on android. The purpose of this game is multiple teams consisting of several people going four rounds at a minute each. When a picture is pulled from a library of images one team guesses till time runs out. If not answered each opposing team member gets to answer till 15 seconds runs out.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

Requires the involvement of one or more teams.

Each team comprises a specific number of individuals.

Distinctiveness is necessary for both game and team names to allow users to verify name availability.

The game operates in a singular instance only.

Compatible with a variety of platforms.

These criteria are essential when developing software and code. Despite being a gaming-related concern, application development must be taken into account. The objective for The Gaming Room is to ensure functionality across all hardware. While currently accessible on Android, integration into another mobile platform is required. Additionally, compatibility with devices such as Windows, Linux, and Apple is crucial. To achieve this, one must explore options like rewriting the code in Swift for Apple devices or finding ways to utilize existing code that can run on different platforms by incorporating other programming languages. This approach harnesses the strength of multiple languages to create robust code.

## [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)

Entities establish relationships among the Game, Team, and Player classes. In simpler terms, all of them either receive information from the Entity or inherit it. This relationship can be visually represented using inheritance in UML, illustrating the sharing of common attributes like "name" and "id" across each class, indicating a superclass for Entity. Upon closer inspection, the relationship between Team and Player is characterized as a "has a" type, while GameService incorporates Games and a Team within Game. In UML, this is described as aggregation (HAS-A). When we say a user "has a," it generally means they possess both an instance of one class and a reference to another class's instance. Examining the diagram reveals that GameService holds references to Games, Games has references to Teams, and Teams have references to Players.

**"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 can be used as a server platform, and macOS Server is available for hosting web applications.  Licensing costs for macOS Server may apply.  Consider using popular web server software like Apache or Nginx on macOS for web hosting. | Linux is a popular choice for web server hosting due to its stability, scalability, and cost-effectiveness. Many web hosting solutions are based on Linux, such as Apache, Nginx, and various cloud services. Licensing costs are usually low or non-existent, making it an attractive option. | Windows Server is another viable option, offering good support for hosting web applications. However, it typically comes with licensing costs, which can vary based on the specific edition and configuration. | Host the backend web application on a server platform (e.g., Linux) and ensure compatibility with iOS communication protocols.  Use tools like Xcode Server for continuous integration and delivery. Host the backend web application on a server platform (e.g., Linux) and ensure compatibility with Android communication protocols.  Integrate with Google Cloud services for scalability and reliability. |
| **Client Side** | For Mac desktop clients, ensure compatibility with Safari, the default browser.  Use Xcode for native macOS app development, which requires proficiency in Swift and Objective-C.  Implement responsive web design for web-based interfaces to adapt to various Mac screen sizes. | Ensure compatibility with popular web browsers on Linux, such as Firefox and Chrome.  Develop web-based interfaces that are responsive to different Linux desktop environments.  Linux mobile clients are less common but can run web-based applications using web browsers. | Ensure compatibility with Microsoft Edge for Windows desktop clients.  Use Visual Studio for native Windows app development (UWP).  Implement responsive web design for web-based interfaces to accommodate different Windows screen sizes. | Develop native iOS apps using Xcode and the Swift or Objective-C programming languages.  Ensure compliance with Apple's App Store guidelines for distribution. Develop native Android apps using Android Studio and Java/Kotlin.  Ensure compliance with Google Play Store guidelines for distribution. |
| **Development Tools** | Xcode (macOS):  Primary integrated development environment (IDE) for macOS and iOS development.  Supports Swift and Objective-C for building iOS/macOS applications.  Offers features like code editing, debugging, and Interface Builder for designing user interfaces.  Visual Studio Code (Cross-Platform):  Lightweight and extensible code editor compatible with macOS, Linux, and Windows.  Supports numerous programming languages and has a rich ecosystem of extensions.  Suitable for web development, scripting, and general-purpose coding.  Homebrew (macOS):  A package manager specifically for macOS.  Facilitates the easy installation of development tools, libraries, and software packages on macOS. | Visual Studio Code (Cross-Platform):  Versatile code editor compatible with macOS, Linux, and Windows.  Offers extensive language support, extensions, and a thriving developer community.  Eclipse (Cross-Platform):  Open-source integrated development environment (IDE) suitable for multiple programming languages.  Known for Java development but supports other languages via plugins.  Used for software development across various domains.  GCC (GNU Compiler Collection):  A collection of compilers for languages like C, C++, and Fortran.  Essential for low-level system development and native Linux software development. | Visual Studio (Windows):  Microsoft's integrated development environment (IDE) for Windows applications, web development, and cloud services.  Supports languages like C#, C++, and .NET.  Offers debugging, testing, and profiling tools.  Visual Studio Code (Cross-Platform):  Lightweight, versatile code editor available on Windows, macOS, and Linux.  Extensively used by developers for cross-platform development and offers a wide range of extensions.  JetBrains IntelliJ IDEA (Cross-Platform):  A robust IDE for Java, Kotlin, and other JVM-based languages.  Provides advanced code analysis, debugging, and tools for web and mobile development.  Microsoft SQL Server Management Studio (Windows):  Designed for database development and management on the Windows platform.  Equipped with tools for querying, designing databases, and managing SQL Server instances.  Git (Cross-Platform):  A distributed version control system used for source code management.  Available on Windows, Linux, and macOS, crucial for collaborative development.  Docker (Cross-Platform):  A containerization platform for packaging and distributing applications across different environments.  Supports consistent development and deployment on Windows, Linux, and macOS. | Cross-platform development frameworks like React Native or Flutter can streamline development for both iOS and Android with shared codebase.  Licensing Costs:  Licensing costs for development tools (e.g., Xcode, Android Studio) may apply, and distribution on app stores (Apple App Store, Google Play Store) often involves registration fees and revenue-sharing agreements.  Overall, supporting these platforms requires careful consideration of development tools, platform-specific requirements, and compatibility testing to ensure a seamless experience for users on Mac, Windows, Linux, and 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:

1. **Operating Platform:** To ensure scalability and reliability, The Gaming Room should implement a server operating system, and a highly recommended choice is Microsoft Windows Server 2016. This operating system is robust and flexible, capable of hosting a diverse range of applications, such as Draw It or Lose It, and it offers excellent scalability to easily adapt to the growing needs of The Gaming Room.
2. **Operating Systems Architectures:** Windows Server 2016 follows a multi-layered architecture consisting of key components:

Kernel: At the core, the kernel manages hardware and provides fundamental services to other layers.

User mode: This layer isolates applications, providing a safeguard against malicious code by separating it from the kernel mode.

Drivers: Acting as an intermediary, drivers facilitate communication between the operating system and hardware, abstracting the hardware details.

1. **Storage Management:** The Windows Server 2016 operating system supports various storage devices like disks, tapes, and optical discs. It also incorporates diverse storage management techniques, including RAID, volume shadow copies, and deduplication.
2. **Memory Management:** Employing various techniques such as paging, memory mapping, and virtual memory, Windows Server 2016 manages memory effectively to track usage and ensure applications have the necessary memory access.
3. **Distributed Systems and Networks:** Designed to support distributed systems and networks, Windows Server 2016 offers features for seamless development and deployment of distributed applications, including: Remote Procedure Calls (RPCs): Facilitating communication between applications over a network. Distributed File Systems (DFS): Enabling access to files stored on remote servers. Web Services: Allowing applications to communicate using the web.
4. **Security:** Windows Server 2016 boasts an extensive set of security features to safeguard data and applications, incorporating: User Accounts and Passwords: Essential for user authentication and access control. Data Encryption: Protects sensitive data from unauthorized access through encryption. Firewalls: Prevents unauthorized network access. Intrusion Detection Systems (IDSs): Detects and alerts administrators to potential security threats.