

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## Table of Contents

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

[Table of Contents](#bookmark1)**2**

[Document Revision History](#bookmark2)**2**

[Executive Summary](#bookmark3)**3**

[Requirements](#bookmark4)**3**

[Design Constraints](#bookmark5)**3**

[System Architecture View](#bookmark6)**3**

[Domain Model](#bookmark7)**4**

[Evaluation](#bookmark8)**5-7**

[Recommendations](#bookmark9)**8-9**

## [Document Revision History](#bookmark10)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/24/2024 | Connor Bailey | Initial Commit |
| 2.0 | 10/06/2024 | Connor Bailey | Added Evaluation section of documentation |
| 3.0 | 10/20/2024 | Connor Bailey | Added Recommendations section of documentation |

**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](#bookmark11)

The Gaming Room is developing a web app of their existing Android game *Draw It or Lose It* to expand the platform’s reach across multiple devices. This design document outlines the plan for implementing the game in a distributed web environment. The end product should support multiplayer teams across different platforms and enforce a single instance of the game in memory. Our team aims to provide a scalable and efficient solution that can be easily maintained.

## Requirements

1. Cross Platform: The game must be accessible via web browsers and the current Android app.
2. The game must support multiple teams participating simultaneously.
3. Unique Team/Player Identifiers: The game must allow teams and players to select unique names, and a system must exist that validates and enforces name uniqueness.
4. Single Game Instance: Only one instance of the game can run in memory at any given time.

## [Design Constraints](#bookmark12)

1. Platform Compatibility: Since the game needs to run on various web browsers and operating systems, the project must account for compatibility issues between the different platforms.
2. Network Latency: Web apps can be limited by the network conditions they are being ran on. The game must be designed to handle network delays without affecting the user experience.
3. Player Synchronization: With the multiplayer requirements of the game, in order to ensure fairness of gameplay, the game must ensure that the state of all players is synchronized together to ensure that no one player is gaining an unfair advantage.
4. Security Consideration: Given that the game will be accessible online, security precautions should be taken to protect user data and ensure fair gameplay.

## [System Architecture View](#bookmark13)

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](#bookmark14)

At the core of the UML diagram, the GameService class follows the Singleton design pattern and ensures that only one instance of the game service is running at any given time. The GameService manages a list of games and provides public methods for adding games, retrieving games by name or game ID, and generating unique IDs for teams and players.

The Game class holds a list of teams and represents the individual games. The Teams class contains a list of players and represents one of the teams playing the game. The Player class represents one player on the Team. These three classes all inherit from the base Entity class, which encapsulates attributes like id and name. This demonstrates inheritance and reusability of code by combining shared attributes and behaviors into a base class.

The ProgramDriver class contains the main method that launches and runs the game. Additionally, the SingletonTester class verifies that the Singleton pattern is correctly implemented by testing how the GameService is instantiated. These components of the project show how encapsulation and the singleton design pattern are used to complete the game’s software requirements.

**"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](#bookmark15)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| **Server Side** | MacOS can serve as a host for web based applications but is typically not used for large scale server deployments due to the licensing costs and due to a restriction in hardware. MacOS is known for its stability and strong integration with other Apple services. MacOS is more suitable for small internal applications. | Linux is the preferred platform for hosting web applications due to the open-source design and flexibility. It offers a variety of deployment methods like Apache and Nginx, making the server scalable. Licensing costs are typically non-existent unless paying for extended support. | Windows Server is commonly used in the enterprise for hosting web applications. Licensing Windows, especially in an enterprise setting, can become expensive. However, it provides a familiar user interface for most users and supports a variety of server configurations. This platform is typically used for Windows specific applications. | Mobile devices are not typically used for hosting web applications. This platform lacks the hardware and software infrastructure to support large scale server deployments. Hosting on mobile is not practical for the needs of most enterprise applications. |
| **Client Side** | Developing a web application for MacOS typically involves targeting the Safari web browser using WebKit. Developers need to ensure that the application runs smoothly across different screen resolutions and versions of Safari. The time and cost considerations are considered typical. | Developing web applications for Linux typically involves targeting the Firefox and Chrome web browsers. Linux users typically expect open-source solutions, but this is not required for deployment. The expertise needed for Linux development can sometimes be higher due to the different flavors of Linux distros, but the time and cost considerations are considered typical. | Developing web applications for Windows typically involves targeting Microsoft Edge, along with other browsers like Firefox and Chrome. Development costs for this platform may be slightly higher than normal to accommodate different OS versions and through the need to accommodate Windows features such as touchscreen support. However, Windows has the largest user base of any of the other platforms, making it essential to prioritize compatibility. | Developing web applications for mobile requires careful consideration of design. Typically, the interface on mobile must be drastically changed from the desktop version due to the limitation of screen size. The application must perform efficiently on mobile browsers, or a native app should be created that interfaces with our web server. Mobile frameworks between iOS and Android can differ in implementation, which requires an increased level of expertise, or a varying development team with experts on both platforms. Due to the accessibility of mobile devices, this platform should be considered as essential for wide spread adoption. |
| **Development Tools** | Developing for native Mac applications typically happens in Xcode, but other cross platform IDEs can be used. Swift and Objective-C are used for native application development, but web technologies such as HTML5, CSS, and Javascript are all compatible and available. Licensing costs for these development tools are typically minimal, but there is an associated cost for compatible hardware. | Developing on Linux typically requires tools like Eclipse, Vim, or Visual Studio Code. Linux supports most modern programming languages including Javascript and CSS. The development costs are typically nonexistent on this platform and licensing fees tend to be minimal. | Developing on Windows often requires tools like Visual Studio from Microsoft. Windows supports most modern programming languages, but C# and .NET are commonly used for native applications. Javascript is supported on Windows. Licensing costs for development tools on Windows can be significant, though free versions of the same software do exist with limited functionality. | Developing for mobile devices often rely on with Android Studio (for Android) or Xcode (for iOS). There may be licensing costs for certain development tools and APIs, although this is usually only true for the iOS platform. The main cost and time considerations come in when considering cross platform compatibility and maintaining the same user experience across platforms. |

**Recommendations**

1. **Operating Platform**:

In order to support the expansion of *Draw It or Lose It* to other platforms, I recommend using a Linux based operating system for the back-end infrastructure. Linux operating systems, such as Ubuntu Server and CentOS, offer several advantages that align with the client’s needs. Linux is often regarded for its stability and performance and is a common choice for hosting web applications. Additionally, it is also much more cost effective. Most Linux distributions (including Ubuntu and CentOS) are both free and open source, which means that the deployment and licensing costs for these operating systems are typically free. Linux is also compatible with deployment tools (like Docker) that support containerization technologies. These tools can help to streamline deployment and ensure consistency across different platforms.

2. **Operating Systems Architectures**:

Linux uses a monolithic kernel architecture. In this architecture, the core functions of the operating system (like memory management, file management, and process scheduling) are all handled in kernel space. Having these important functions in the kernel allows for faster communication between system components, and increased reliability of the system as a whole. In addition, the modular nature of the Linux kernel means that services can be loaded dynamically. This ensures that only the necessary components are running at any given time and leads to better resource utilization. Linux’s support for multi-threading and process isolation also ensure that many different instances of the game can be running at the same time without interfering with one another.

3. **Storage Management**:

For *Draw It or Lose It,* I believe a cloud-based storage solution, such as Amazon’s S3 object storage, is the most suitable option for handling the game’s storage needs. A cloud storage solution provides scalability that will allow The Gaming Room to expand the game’s storage capacity as their player base grows without any concerns over physical storage limitations. These services are also highly available and redundant, which ensures that the player data, game sessions, and team information are always accessible to users. Cloud providers can also offer features such as automatic backups that help to safeguard this data and ensure that it could be recovered in the event that there is a disruption to the service. This cloud-based solution also easily integrates with Linux to provide low latency access to data. This means that processes that update the game state and player progress are very quick and responsive, which is crucial for user retention and smooth gameplay. There are also additional security considerations when using a cloud based storage solution, like encryption, that can enhance user privacy and fair gameplay.

4. **Memory Management**:

In a Linux based operating system, one of the core memory management techniques that would be well suited for *Draw It or Lose It* would be virtual memory. This feature allows the Linux system to use both the physical RAM and the disk space (also known as swap space) to manage large applications and multiple users at the same time. This ensures that even as the player base increases, the game can maintain high performance without exhausting all the system’s resources. Additionally, Linux uses another feature called memory paging. This feature works by breaking down processes into smaller “pages”, which can be moved between memory and storage as needed. This prevents stale data from taking up memory as users arrive and leave the game, ensuring that the back-end stays responsive. Linux also uses demand paging, where memory pages are only loaded into RAM when they are needed, which also helps to reduce stale data in memory.

5. **Distributed Systems and Networks**:

In order to make *Draw It or Lose It* cross-platform, the game must be designed as a distributed system. In this system, the game logic and data can be split across multiple servers, allowing players on different platforms to interact and play together seamlessly. For this game, a client-server model is ideal. In this model, the game clients will connect to a central server that manages the game state and player interactions. The network must support low-latency to ensure that the data synchronization between clients happens in near real time to provide a smooth and enjoyable gameplay experience. Web technologies like Web Sockets and HTTP communication can be used for persistent, fast communication between the server and clients and can keep the game state synced for all players.

In terms of network design, the system should include load balancers to distribute traffic evenly across multiple servers. A load balancer helps to avoid bottlenecks during peak times and ensure that the back-end’s resources are being properly utilized. Redundancy should also be incorporated through the use of distributed databases and replication of these databases. This helps to reduce the impact of outages in a distributed system as another server can take over if one ever becomes unavailable.

6. **Security**:

Security considerations in a distributed environment that hosts a multiplayer game is very important to ensure fair gameplay. In Linux, features such as firewall management and access controls would help to protect the game servers from unauthorized access from malicious actors. In order to safeguard user data, implementing encryption at rest and in transit should be considered essential features. Encryption protocols such as TLS (Transport Layer Security) should be used to secure data during communication between the server and the client to prevent interception by malicious actors. User authentication and access control mechanisms should also be implemented to ensure fair gameplay. This will prevent malicious users from accessing the data and player profiles of legitimate users, while also preventing unauthorized modification to entries in the game’s database.