

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 4**

[**Domain Model 4**](#_Toc115077324)

**Evaluation 5**

**Recommendations 8**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 1/23/25 | Daniel Meza | Make and design for the game development based on the clients’ requirements and developing requirements. |

**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)

Our new client The Gaming Room wants to develop a web-based game that serves multiple platforms based on their game Draw It or Lose It. The software design problem is developing a multi-team game system where every team has multiple players. The system must manage real-time interactions between the players and must ensure that no two games, teams or players have the same name. To address these problems, making a web-based, server-client where the server manages the game logic, handles real time communication and ensures data storage.

## Requirements

Business Requirements:

1. Multiple teams: The game requires one or more teams playing at once. Each team will compete in the game and the system should allow the creation of multiple teams with one game session.
2. Team members: Each team will have multiple players. The system should allow for addition and management of players for each team during the game.
3. Unique Team and Game names: The game and team names need to be unique to prevent issues. The user needs to be able to check the availability of a name before being able to select it.
4. Rounds and Time Limits: The game will have four rounds, each of them lasting one minute. The game has to handle the time limit for guessing efficacy. If the team fails to guess the image in time, the other teams should have one chance to guess the image in 15 seconds.

Technical Requirements:

1. Real- Time Gameplay: The game must support real-time interactions between players. This will require a real-time communication framework.
2. Single instance of a game: Only one instance of a game can exist in memory at any time. The system needs to create and manage unique identifiers for each game, team and player to make sure that there are no issues or duplications.
3. Platform Independence: The game will be web-based so that must be accessible across platforms and devices such as smartphones and computers. It must also be compatible with web browsers such as Chrome, Edge, and Safari.
4. Unique names: When making a new game or team, the system needs to ensure that names are unique and not duplicated. This will require name checking in the database to prevent issues.
5. Interface and User Experience: The user interface must be user-friendly to all devices. This includes instructions, responsive design, and smooth transitions between games.

## [Design Constraints](#_2et92p0)

1. Cross-Platform Compatibility: Since the game is web-based it requires that the game works on different devices such as smartphones, tablets, and computers. The game must be accessible to web browsers such as Chrome, Edge, and Safari.
2. Scalability: The game should be able to scale a growing number of games, teams, and players. As the number of users increases the system must be able to handle more games and players. This requires an effective database.
3. Single Instance of a game: Only one instance of a game should exist in memory at any given time. This requires that each game, team, and player must have unique identifiers to avoid issues.
4. Real-Time Gameplay: The game requires real-time gameplay between players and teams. Since the teams will be guessing the image there must be minimal latency to have a good user experience.

## [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)

ProgramDriver Class: The ProgramDriver class is an entry point to the application, it contains the Main() method that initiates the program execution. It uses the SingletonTester class, which means that it will interact with it to use the singleton pattern. The <<uses>> shows the dependency.

SingletonTester Class: The SingletonTester class is used to verify that the Singleton pattern is implemented correctly in the system. It will test that only one instance of the GameService class is created.

Entity Class: The Entity class is a base class for the other classes in the system such as Game, Team and Player. It has sharded attributes and methods that the derived classes will inherit. The attributes are id(long) a unique identifier for the entity for the Game, Team and Player. The methods are constructors and getters for id and name. The toString() is a way to represent the entity as a string.

GameService Class: The GameService class is used for managing other game operations. It manages Games, Teams, and player. It implements the Singleton pattern making sure that only one instance exists throughout the system. The game attribute is for a list of active games. nextGameId, nextPlayerId and nextTeamId are unique identifiers used to create new games, team, and players objects. The getInstance() method returns the single instance of GameService after the Singleton design. addGame() adds a new game to the system. getGame() and getGameCount() gets the games by the Id or name, and count how many games exist. The gameNextPlayerId and getNextTeamId get the next available unique Id for a team or player.

Game Class: The Game class is used to manage a game and the list of teams. The Teams attribute is for a list of teams that are participating in the game. addTeam() adds a new team to the game. toString() is a string representation of the game. The Game class has a relationship with the Team Class represented by a solid line and 0…\* this signifies that the game can contain zero or more teams.

Team Class: The Team Class is for a team in a game it contains a list of players. It also allows to add and manage players in a team. The player attribute is a list of players that belong to a team. The addPlayer method adds a player to a team. toString() is a string representation of the team. The Team class has a relationship with the Player class represented by a solid line and 0…\* this signifies that the game can contain zero or more players.

Player Class: The Player class is used for an individual player in a team. It has a unique Id and name of the player. The id attribute is for the players unique identifier and name is for the name of the player. toString() is a string representation of the player. The Player class inherits from the Entity class sharing common properties such as id and name.

**"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** | macOS has limited software and hardware capabilities for a server. Macs are typically more suited for development rather than servers. It does have great stability for web development tools. There is no extra cost for macOS on a small-scale, but it can cost more when scaling up when relying on macOS servers. | Linux is great for web hosting because of its stability and performance. It also has good security features, and it can also be configured for high availability and load balancing. Linux is used in production environments and supports cloud services. Lunix is open-source and free making it cost effective. | Windows has great resources for web hosting with different technologies such as .NET, which can be great for web applications. Sometimes these technologies require fees to use them. Windows server editions are used in enterprise environments, but it can also require more resources. Windows can have licensing fees for the server operating system. | Mobile devices are not suitable hosting web applications. They do not have the proper hardware and software for certain tasks. They are usually the client-side devices and rely on cloud services. Mobile devices do not apply here since they do not host. |
| **Client Side** | Macs have a great user interface and provide a good web environment for browsers such as for Safari. The weakness with mac is that iOS is not cross compatible with other operating systems such as Windows and Linux. | Since not many people use Lunix it can be hard for people to learn the Linux systems. It can have a more challenging development for cross platform apps, but the good thing is that it can be configured to work with the most common browsers. | Windows is commonly used, and web applications are usually compatible across browsers such as Chrome, and Edge. It is also required to do extra testing for mobile compatibility. | Mobile devices require certain design and development. iOS and Android also need more time for testing to make sure they work on all screens and OS versions. |
| **Development Tools** | macOS uses Xcode in iOS development and has many tools for web development. Such as VS Code, Terminal, and Sublime Text. Xcode is exclusive to macOS so that will limit the cross-platform development. The developing team for mac will require expertise in iOS and macOS development. If a cross-platform development is required, then there will need to be a separate team or a framework such as Flutter. Xcode is free, but macOS comes with licensing cost when used for commercial purposes. Other development tools may require licensing fees. | Lunix uses a lot of open-source tools for web development such as VS Code, Emacs and Vim. It supports many programming languages like Python, JavaScript and PHP. It also supports frameworks like Node.js and Django. The developing team will need to have experience in server management and web frameworks. This also requires team expertise in Linux administration. Most of the developing tools on Linux are open-source and free but paid tools will require fees. | Windows offers a lot of development tools for web and desktop application development. These include Visual Studio, Eclipse and Unity. These offer many programming languages such as C#, C++, and JavaScript. They can be used for desktops and web development. The developing team will need experience in Microsoft technologies and be familiar with Windows Server for hosting the web application. Visual studio is free but certain editions require licensing cost and Windows servers may also add additional fees. | For iOS, Xcode is required to make apps. Android normally uses Java to make apps. Using cross platform frameworks such as Flutter and React Native Developing for mobile requires more attention to detail for specific user interface depending on what device is being used. The developing team for mobile will require experience in iOS development and Android development. Development for cross-platform will require additional expertise with Flutter or React Native. iOS Xcode is free, but Apple Developer Program has an annual fee. Android requires fees for development on the Google Play Store. |

## 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**: I recommend using Linux as the operating system for hosting the application because it is open-source, flexible and cost effective. It has the largest amount of cross-platform support while being able to scale and run on different devices such as computers, smartphones and tablets. Linux also has great stability, performance and security, which is important for real time multiplayer applications. For the client-side application, the web-based game is easily accessible from different platforms and devices. Using Linux for the server side allows for better resource management, load balancing, and handling multiple players, which is important for better gameplay.
2. **Operating Systems Architectures**: Linux uses monolithic kernel architecture. Most of the operating system services such as memory management and networking are handled by the kernel itself. This architecture helps provide efficiency and scalability, which allows Linux to handle high workloads effectively and allows multiple users to access the game at the same time without issues. This is important for real time gameplay in DRAW IT or LOSE IT. This architecture also supports multithreading and multiprocessing, which allows it to scale dynamically as the number of players increases.
3. **Storage Management**: Cloud storage will be a great choice for storage management. Cloud storage services such as AWS S3, Microsoft Azure Blob storage and Google Cloud Storage are great options because they are all flexible, scalable and reliable. This will allow game data, player information and game assets such as images to be stored securely and accessed quickly from a device. Cloud storage can be scaled to handle great volume of data as the number of players increase, and it can retrieve large data with minimal latency which is important for real time gameplay.
4. **Memory Management**: Linux has a great memory management system because it uses virtual memory which allows the system to handle more processes than the physical RAM would. This is important for DRAW IT or LOSE IT because it will help by managing multiple players and games without crashing or slowing down. Linux also uses shared memory and memory pools which allows the system to efficiently allocate memory resources to processes that need them the most. This ensures smoother gameplay, especially during peak times.
5. **Distributed Systems and Networks**: WebSocket is a great for communication between the client and server because it enables real-time interactions without constant polling. This allows the game to send and receive real-time updates such as showing the images and when players are guessing what the image is with minimal latency without putting unnecessary strain on the server. The game’s server should be load balanced using tools such as Azure Load Balancer and Nginx to manage high usage and prevent downtime.
6. **Security**: To make sure user data is protected I recommend using SSL/TLS encryption to secure all communication between the game client and server. This will protect sensitive data such as player credentials and game information. Also using OAuth 2.0 should be used for user authentication to securely manage player accounts. This will make sure only authorized users can access their data and game session. The game data should be encrypted both when being stored and transmitted, this will safeguard against unauthorized access and ensure data privacy regulations are being met.