

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

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

[**Table of Contents 2**](#_30j0zll)

[**Document Revision History 2**](#_3znysh7)

[**Executive Summary 3**](#_2et92p0)

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

[**Design Constraints 3**](#_1t3h5sf)

[**System Architecture View 3**](#_4d34og8)

[**Domain Model 3**](#_2s8eyo1)

[**Evaluation 4**](#_17dp8vu)

[**Recommendations 5**](#_26in1rg)

## [Document Revision History](#_lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0  1.1 | 03/24/24  4/7/24 | Wayne Silva  Wayne Silv | Design template for the first iteration of Draw It or Lose It.  Added an assessment of operating platforms for the purpose of creating cross platform applications |

**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](#_35nkun2)

The Gaming Room has a game by the name Draw It or Lose It that it wants to turn into a web-based game. This game currently runs on Android but the company wants to reach a wider audience. Draw it or lose it is a game where users compete to guess what phrase, title, or thing was selected based on image drawings. The game has four rounds each lasting one minute. Drawings will be complete thirty seconds from when they were started but if no one guesses the opposing team will have 15 seconds to guess.

The web-based game must be multiplatform so that The Gaming Room can expand its user base.

The application will be built utilizing the OOP paradigm and the Java Language. There will be 6 main classes (7 including a tester class). ProgramDriver, GameService, Entity, Game, Team, and Player are the main classes that make up this program. Program Driver will contain the main function and will be the starting point of the application. GameService is a singleton class which will keep track of state pertaining to the creation and loading of new and existing games. It will also track the unique identifiers for nextGameId, nextPlayerId, and nextTeamId. These fields exist to track unique identifiers for their respective classes so that newly created classes each have unique identifiers. When a new instance of a Game, Team, or Player class must be created, GameService will handle the updating of each unique id for the respective class. The Game, Team, and Player class will also inherit from the superclass Entity which contains the private id and name fields as well as public getter methods to retrieve those fields. The Entity class is a polymorphic superclass which is needed to instantiate the subclasses Game, Team, and Player with the necessary methods and fields. Singleton Tester exists for testing purposes. This class will allow developers to test the functionality of the singleton class.

## Requirements

1. *A game instance will be able to have one or more teams.*
2. *Each team will have multiple players assigned.*
3. *Both game and team names will be unique so that players can select an unused name and so that games and teams can be searched by name.*
4. *Only one game instance can exist at any given time.*
5. *Only one instance of GameService can exist.*
6. *Must be multi-platform*
7. *Must be web-based*

## [Design Constraints](#_1ksv4uv)

Since the game application must be web-based, the languages and/or frameworks used must be executable in a purely web-based context. The application must be built using a language or framework that is purpose built for web application development. Javascript, React-Native, and Django are some examples of the languages and frameworks that could be used to develop this game. It is additionally important that the language and tools chosen run on a variety of devices. Kotlin for instance would not be a good choice for this project since it is an Android only development language. Since the game will persist in a distributed environment, care will need to be taken so that game clients communicate effectively in an asynchronous manner. Many players will be playing at once and submitting requests to the server, because of this the game must be able to handle these incoming requests so as to not compromise the integrity of the game or its core logic.

## [System Architecture View](#_44sinio)

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](#_2jxsxqh)

The UML Diagram below is a class diagram that outlines the relationships and overall structure of the classes used in this application. The ProgramDriver is the main class and is associated with the SingletonTester which it makes use of for the purpose of testing. The GameService class is associated with the Game class in a zero to many relationship. There can be 0 games or 100+. This is a singleton class, as such only one can exist in memory at any given time. The Game class is associated with the Team class in a zero to many relationship. Also, the Team class is associated with the Player class in a zero to many relationship. Every instance of the Game class will store a list of the teams in that game and can have more teams added using AddTeam(). Similarly, instances of the Team class store all current players on that team in an array list. The addPlayer() method allows players to be added to the players array list. The Game, Team, and Player classes also inherit from the Entity class. The fields and methods of the Entity class are inherited by its descendants. An example of abstraction in this diagram is the use of private fields and public methods that can be called in different execution contexts in order to set or mutate the values within the private fields. Data inside of the class is not available to the user, except through the public methods. Encapsulation can be observed in the use of classes that contain all relevant contextual data such as their variables and methods. Polymorphism is evident in the relationship by the entity class and its direct descendants. An Entity can be represented as a Game, Team, or Player and it will retain its methods and fields, additionally gaining the new methods and fields of the respective subclass.

**"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](#_z337ya)

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** | OSX is a proprietary operating system with a powerful command line. Versions of OSX exist on both mobile and PC. It can only be run on Apple devices. The server edition has been discontinued. OSX would not be a great choice as a server operating system. | Linux is an open-source OS with a powerful command line and sensible file structure. Linux is a very popular server os. Linux can be run on almost any type of hardware. Desktop and server editions exist for many popular distributions and all are fairly lightweight. | Windows is a proprietary operating system that can be run on a wide range of hardware. Typically, it is run on PCs. The operating system is not free. Windows has various versions including server editions. | <Evaluate Mobile Mobile devices are mainly meant for client side applications and do not make the best servers. These devices have much lower system resources, batteries that can be quickly drained, and very few ports for peripherals. They also do not typically have ports for networking. Mobile platforms would be a poor choice as a server. |
| **Client Side** | Development on OSX is quite popular. Many tools exist such as Xcode and Apple have their own languages such as swift, Qt, and objective-C. Qt is cross-platform. Costs are higher than average since proprietary hardware, software, and the developer license are needed. Expertise is limited to those skilled in the OSX ecosystem. | Variations GNU/Linux are fairly common on mobile devices and some desktop environments. The cost associated with development is low. Expertise is better than average in recent years with Linux becoming more popular on the server-side and mobile devices. | Windows clients are incredibly common especially on the desktop. They are less common in mobile devices, which is how many people play games. Due to the popularity of the OS expertise should not be difficult to find and development time should be reasonable. Windows is proprietary so development costs will have to account for testing environments and proprietary tooling. | Mobile Devices are incredibly popular and are how many users engage with games and applications nowadays. Expertise in this space is easy to come by and development time should be reasonable. Many tools exist to emulate mobile devices environments for testing purposes. |
| **Development Tools** | Development tools and languages include Xcode, Swift, Objective-C, React-Native and Qt. | Linux has many tools and languages available. Common languages used in development are Python, Javascript, and React-Native. | Common tools such as Visual Studio are widely used. ASP.Net is commonly used to build web apps. C#, Javascript, React-Native, and Xamarin are used for cross platform development as well. Azure cloud is a major selling point as a top tier cloud computing platform. | Kotlin, Flutter, Javascript, NodeJS, Python and man other languages exist to build cross platform apps on mobile. It is easy to emulate a wide range of devices using platform specific tools such as BlueStacks or XCode. |

## 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**: Using Linux as the team's operating platform will allow them to keep costs low and have plenty of tooling available. Linux is fast, lightweight, and free (in most cases). It makes for a powerful server and can serve as the platform for a flutter or kotlin web-based game that would be easily deployable to many platforms. Linux allows the team to scale as needed and they can even move into the cloud utilizing containerization of their linux environment.
2. **Operating Systems Architectures**: Linux uses an open-source kernel with process scheduler and robust virtual file system. The linux file system has directories such as /www that can be used for hosting parts of the game service. The linux process scheduler is also powerful and allows users to set up cron jobs such as running game processes on startup and reboot. This will be important in case of server crashes or during updates.
3. **Storage Management**: The most appropriate storage management system is a solid state drive with plenty of storage capacity. This medium will allow very fast reads reducing load times for the end user. While more expensive than hard discs, the read/write speeds of an SSD is superior for this use case.
4. **Memory Management**: Memory management with Linux is powerful. Many settings exist in /proc that can be used to tailor the system to our team’s needs.
5. **Distributed Systems and Networks**: Because Draw it or lose it will communicate with many different platforms using an API on the linux server that accepts https requests will be critical. Https is a low level protocol that most if not all computing platforms can understand. Allowing the various clients to send requests such as GET and POST and having the appropriate API endpoints on the server that will handle these requests will allow our game developers to encapsulate game logic into the respective endpoints. For instance if a user wants to make a guess they can make a GET request to <http://drawitorloseit.com/guess> passing in their guess to the server. In response the server can send back a response that lets them know if they were correct or not and update the game state as needed.
6. **Security**: <Security is a must-have for the client. Explain how to protect user information on and between various platforms. Consider the user protection and security capabilities of the recommended operating platform.> Linux has robust security features such as encryption, ssh, top notch access control, and permission structures. We can leverage these features to ensure that only privileged users have access to sensitive processes and actions. For instance, by setting up a hierarchical user structure, we can assign developers, techs, admins, and anyone else that needs access to the server the appropriate level of permissions and access. On the client side we will need to make sure our game has the appropriate access and permissions within a domain of protection across various platforms.