

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 1.1

## 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 | 01/28/2024 | Mark Turner | 1.0 03/18/2023 Jeffrey Sparks Changes have been made to the cover page, the  document revision history, the executive summary,  design constraints, system architecture view, domain  model, and recommendation.  Added short executive summary, list of design constraints and requirements, high-level overview of domain model, evaluation of various operating systems, and recommendations for software used for the application’s infrastructure. |
| 1.1 | 2/11/2024 | Mark Turner | Updated overview of operating platforms for the developer tools, client, and server-side engineering options |
| 1.2 | 2/25/2024 | Mark Turner | Updated recommendations section | |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room wants to develop a web-based, multi-platform version of their existing game, ‘Draw it or Lose It’, which is currently only available for Android devices. In ‘Draw it or lose it,’ teams of players compete to guess what phrase, title, or object a drawn image represents. The application renders images from a large library of stock drawings as clues and prompts a specified team to guess what the image represents during the rounds of a game. Drawings are steadily rendered and completed at the 30 second mark of a round within a game. Each game consists of four 1-minute rounds in which the players on a selected team will be prompted to guess the puzzle before the round is over. If the chosen team does not solve the puzzle before the round’s time expires, the remaining teams will each have a 15-second opportunity to offer a single guess and solve the puzzle.

## Requirements

* Application must be compatible with multiple computing platforms including commonly used desktop and mobile operating systems.
* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

The following design constraints can be gleaned from the client’s requirements for the application:

* The application must support core functionality across multiple operating systems.
* Game instances cannot be duplicated but may potentially be shared across platforms.
* The application must support the core game functionality and provide a consistent, seamless experience to users regardless of their operating platform.
* Game and team names are unique across operating platforms.

## [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)

As depicted in the diagram below, GameService class utilizes the Singleton design pattern so that only one instance is present within the application at any given time after it’s initialization. This helps coordinate sharing global resources across modules within the application. GameService shares a “0 to many” relationship with the Game class via a property called “games” which is an array of Game objects. The Game, Team, and Player child classes all inherit from a common parent “Entity” class which contains common methods shared between each class. The Game class has a “0 to many” relationship with the Team class by containing a ‘teams’ property which is a list of Team objects. Similarly, the Team class shares a “0 to many” relationship with the Player class in the same fashion with its “players” property.

The ProgramDriver is responsible for orchestrating the application’s functionality via the Main method whilst making use of the SingletonTester class for basic verification of its internal functionality.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Apple’s MacOS provides stable, user-friendly environments through intuitive graphical interfaces and robust operational features. Additionally, custom user-experiences and performance optimizations can be engineered specifically for devices running on Apple’s software. However, operational/hosting costs, licensing and customization limitations for both hardware and software can introduce additional design constraints or quickly surpass the budget restrictions outlined by the client. | Linux is a widely used, highly customizable, extremely versatile option for hosting web-based applications. In lieu of enterprise level support, Linux has a large, active community of supporters who help to develop and optimize certain distributions like Ubuntu. Linux is free and is compatible with a large variety of hardware solutions, making it easy to design for scalable web applications while remaining under budget. However, Linux distributions aren’t very user-friendly and rely heavily on the use of a command line. | Windows provides robust, user-friendly environments through intuitive graphical interfaces and robust operational features. Additionally, custom user-experiences and performance optimizations can be engineered specifically for devices running on Microsoft’s software. Windows supports a wide variety of hardware solution, making it easy to design for scalable web applications. However, Windows server solutions can be costly, often requiring a heavy development and financial investment to rely on Microsoft’s systems which often require paid licenses. | Mobile devices are traditionally used for client-side interactions with a focus in optimizing and simplifying the user’s experience. Whilst ubiquitous in the modern world, they have limited hardware capabilities and various caveats depending on the manufacturer. This can make it challenging to scale web applications appropriately and impractical to maintain. Although entirely possible, it can be costly to invest in a serious hardware solution using mobile devices as servers. Additionally, it may cost additional development time to create an effective server-based deployment method with mobile devices as the operating platform as they are typically used as clients and do not offer in-built deployment functionality. |
| **Client Side** | Supporting Mac clients requires expertise with MacOS and understanding its limitations in terms of security, customization, and supported runtimes. Experience with the swift programming language is necessary to create a native experience to Mac users that remains consistent across operating platforms. It can be costly to create these experiences for Mac clients as only Apple devices are configured to support the MacOS and are needed for software development and testing. | Supporting Linux clients requires experience with commonly used Linux distributions and understanding their limitations in user intractability, security, and programming paradigms. While cost-effective, it can take time to design a consistent multi-distribution user experience that supports the basic functionality of the client’s application requirements. | Supporting Windows clients requires expertise with the Windows OS and its limitations in terms of security, customization, supported runtimes, and general knowledge of the Microsoft ecosystem. Developers with expertise designing .NET applications would be best suited to understanding these design constraints. Additionally, it may take time to seamlessly integrate the application with the Microsoft ecosystem to explore options like Single sign on. | Supporting clients using Mobile devices requires extensive expertise with developing portable, scalable, and secure applications for commonly used mobile operating platforms like iOS and Android. The programming paradigms between each OS can differ greatly and multiple development teams may be needed to ensure a consistent client experience is provided regardless of operating platform. Additionally, mobile devices have greater hardware limitations than desktops, meaning the application will need to utilize efficient and potentially external storage techniques. |
| **Development Tools** | Common development tools and programming languages for MacOS include Homebrew, Git, Xcode, Visual Studio, Swift, Objective-C, Docker, VirtualBox, etc. Save visual studio, the majority tools can be used at an enterprise level without a paid license. | Common development tools and programming languages for Linux include Python, C, C++ Visual Studio Code, Java, Visual Studio Code, Kuberenetes (K8s), etc. Save visual studio, these tools can be used at an enterprise level without a paid license. While the k8s software is free, it costs money to deploy and manage applications using cloud infrastructure. | Common development tools and programming languages for Windows include Git, C#, Visual Studio, Java, Visual Studio Code, Powershell, VirtualBox, etc. Save visual studio, these tools can be used at an enterprise level without a paid license. Additionally, depending on the necessity of the developers, the full suite of Windows OS features requires a paid license. | Common development tools and programming languages for Mobile devices include Git, React, Angular, Kotlin, Swift, Visual Studio, CircleCi, Electron, etc. Save visual studio, the majority of these tools can be used at an enterprise level without a paid license. |

## 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**: Due to its versatility, customization, and wide range of development tools, Linux is the recommended operating platform for developing a multi-platform compatible web-based gaming application. It’s a reliable, cost-effective solution that will allow for scalability and alterations/extensions in the design of applications.
2. **Operating Systems Architectures**: Despite its various distributions, the general architecture of the Linux Operating system remains somewhat similar, consisting of a ‘bottom layer’ Kernel responsible for communicating with the underlying hardware and command-line interpreter known as the ‘Shell’ as the ‘middle layer’ which provides an interface for the user to interact with the operating system. Lastly, the top layer of the Linux platform is the Application layer which contains user-friendly applications such as web browsers, IDEs, and text editors.
3. **Storage Management**: While completely possible, it can be impractical and lead to excess costs to properly maintain storage within on-prem server or file system. It’s recommended to use a cloud-based storage system to maintain data for this type of application. Games typically have a lot of image files, unstructured non-relational user data, and other assets that need to be frequently accessed, solutions like DynamoDB and AWS S3 would be a cost-effective way to remedy this problem.
4. **Memory Management**: For Java applications like Draw It or Lose It, the Linux operating platform uses its Kernel to communicate with the Java Virtual Machine to handle the dynamic allocation and deallocation of memory. The Java Virtual Machine is responsible for the memory on the Heap and the execution of the Java application while the Linux kernel helps maintain the concurrent thread execution and physical memory.
5. **Distributed Systems and Networks**: Edge computing solutions and various distributed content delivery networks like AWS Cloudfront provide cost-effective ways to easily distribute the application and its various assets across the internet and reduce load times, improving the overall user experience. Additionally, networking techniques like peer-to-peer networking can help reduce latency between users and improve performance on the server-side by packaging groups of requests.
6. **Security**: When communicating with client devices, databases, and external services, data will need to be encrypted in transit using secure protocols like HTTPS and SSH. Requiring a username and password for each user to access the application or communication through a dedicated secure channel and firewall (configurable on Linux via tools like *iptables*) can help prevent data breaches and external actors from acquiring user data. Multi-factor authentication can be implemented as an additional optional layer of security for end-users and made mandatory for those with admin/developer access to the application.