

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 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0  1.1 | 01/26/25  02/09/25 | Seth Sharp  Seth Sharp | Added:  -Executive Summary  -Requirements  -Design Constraints  -Domain Model Description  Added: |

**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)

Draw It or Lose It is a turn-based game in which teams compete to provide an answer as input that relates to an image rendered during the round. The software requirements include instantiated games needing a minimum of one team to start, each team needing more than one player, the game and team names must be unique, and only one game instance can exist in memory at one time. To accommodate these requirements a singleton design pattern will need to be utilized that includes unique game and team names that can be referenced and searched.

## Requirements

***Business Requirements:***

-Should be similar to “Win, Lose, or Draw”

-Each game consists of four rounds, each lasting one minute

-Drawings are steadily rendered and complete at 30 seconds from a stock library

-Each game needs at least one team, and each team needs more than one player

-Teams guess the puzzle before time expires, and if unsuccessful the remaining

teams have 15 seconds to input a guess

-Scores should be tracked and updated

***Technical Requirements:***

-Game needs to be a web-based application

-Steady image rendering is crucial for improved UX

-Utilize real-time updates for drawing progress and guesses

-Must handle multiple concurrent users and easily scale with demand

-Compatibility across devices and display types

## [Design Constraints](#_2et92p0)

Due to the web-based design and reliance on real-time updates for drawing progress and guesses, network latency needs to be considered regarding UX. Additionally, the handling of multiple concurrent users requires scalable infrastructure that can manage different load demands on the system. This will ensure high levels of responsiveness and efficiency for users within the game, which would also benefit compatibility. Compatibility needs to be addressed and tested directly to ensure proper execution and data consistency across devices. Finally, resource management should be implemented to efficiently manage server resources for image rendering and real-time updates without performance issues.

## [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)

From the UML diagram, the following information can be gathered. Within the GamingRoom application, there exist 7 classes ProgramDriver, SingletonTester, Entity, GameService, Game, Team, and Player. ProgramDriver contains the main method that accesses the functionality of the various classes, and SingletonTester is inherited and used by ProgramDriver to test the singleton design pattern, as dictated by the solid line and arrowhead from the latter to the former denoted “uses”. Continuing, the hollow arrowhead and solid line indicate an association relationship between the Game, Team, and Player classes and the Entity class. Also, the solid lines denoted “0..\*” indicate zero-to-many dependencies between the GameService, Game, Team, and Player classes, respectively. This notation indicates that each class can have zero or more instances of another that it is connected to. As for the class relationships, the Entity class is a base class introduced to hold common attributes and behaviors, as expressed in the provided materials. This allows the classes within the association relationship to have access to the public methods within the Entity class. The GameService class has access to all the public attributes and methods and contains the main functionality for instantiating objects, as seen in its ability to add games and call functions more meaningfully than other classes.

This demonstrates encapsulation by bundling data, methods, and functions into a single unit, as seen in the classes. The access to some components is restricted by having public and private attributes, denoted “+” and “-”, respectively. This helps to prevent interference and misuse of data from users. Polymorphism is demonstrated because it allows objects of different classes to be treated as objects of a common superclass, as seen in the Entity and GameService classes overloading. The Entity() constructor is overloaded, as well as the addGame() function calls within GameService, indicating polymorphism.

**"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** | Advantages:  -User-friendly interface  -Integration with other Apple products  -Supports Apache for web development  Weaknesses:  -Higher licensing cost  -Less server-specific features  -Less community support | Advantages:  -Highly secure with equal stability  - Supports Apache and MySQL  -Licensing reduced because of open-source nature  Weaknesses:  -Requires more technical expertise  -Compatibility can be an issue for some services | Advantages:  -User-friendly interface  -Supports .NET applications  -Vast documentation exists as a support repository  Weaknesses:  -Higher licensing cost  -More resource intensive | Advantages:  -Accesses web applications through browsers  -Increased versatility for users  Weaknesses:  -Limited processing power that can impact performance  -Inferior storage capabilities adding to performance constraints |
| **Client Side** | Cost:  More expensive than alternative OS options, due to OS specific technical requirements  Time:  More time consuming than alternative OS options, due to specific technical requirements and needed expertise  Expertise:  Proficiency is Swift is essential for industry standard development, though Objective-C is supported. | Cost:  Very cost-effective because open-source development lessens licensing cost  Time:  Expertise and compatibility requirements can make development and testing time consuming  Expertise:  Requires more technical expertise to set up and manage, but supports a variety of languages and tools | Cost:  More expensive than Linux, due to licensing cost differences  Time:  Visual Studio allows for streamlined and quick development, due to the extensive support from Microsoft and the community  Expertise:  Proficiency in C# and .NET development is essential for industry standard development | Cost:  Cross-platform development can reduce costs, as well as expertise in Java, or Kotlin, development  Time:  Widespread use and support of Java and Kotlin applications reduces development times  Expertise:  Proficiency in Kotlin and Java is necessary, along with cross-platform dependencies |
| **Development Tools** | The official Apple IDE is Xcode and supports Swift and Objective-C, though Visual Studio supports macOS development | The main language for Linux is C and the main IDE is Visual Studio. Though as discussed other languages and development tools are supported | The main IDE for Windows is Visual Studio and the main languages are C# and .NET | The main IDEs for Mobile Devices include systems like React Native for cross-platform development , and, for each system independently, Xcode for iOS and Android Studio for Android |

## 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 adopting a cross-platform development approach using a framework like React Native. This supports deployment on multiple platforms, including Mac, Linux, Windows, and mobile devices, ensuring greater reach and flexibility. For server-side operations, Linux seems to be the best option due to its cost-effectiveness, reliability, and open-source support network.
2. **Operating Systems Architectures**: React Native should be utilized for a unified codebase that reduces development costs and efforts while ensuring native performance. The primary language for this development would need to be JavaScript, due to this being the main language of the React framework. At runtime iOS runs Objective-C, or Swift, and Android utilizes Java, or Kotlin. An advantage to using React is its compatibility with Linux-based servers for hosting Node.js and database services while cloud providers such as Microsoft Azure can be used for scalability and availability.
3. **Storage Management**: For storage, I recommend using a cloud-based storage system such as Azure to ensure scalability for storing images, user data, and game progress. It will also allow for rapid data access across platforms and support consistency. Additionally, a database management system can be implemented using MySQL for relational data while leveraging a NoSQL database for flexible storage.
4. **Memory Management**: The operating platforms use various memory management techniques including garbage collection, pooling techniques, and client-side caches that can each be utilized to provide meaningful and rapid functionality. Garbage collection benefits Java and Kotlin systems by preventing memory leaks when properly implemented, while pooling techniques can be use for easy object reuse, like player icons or drawings. Continuing, client-side caches can store temporary progress within the game. Server-side memory can be managed to dynamically allocate resources based on real-time demands using engines like Kubernetes.
5. **Distributed Systems and Networks**: To enable communication between platforms, the implementation of RESTful APIs would allow for an exceptionally efficient exchange between clients and the server. Additionally, using a WebSocket-based approach would create a direct avenue to push real-time updates to connected clients, such as drawing progress. Finally, incorporating load balancers to distribute traffic, and increasing support for some features and functions during outages, will address fault tolerance and dependencies.
6. **Security**: Securing user information across platforms will require the adoption of HTTPS protocols for secure communication, and the use of token-based authentication can aid in increasing security for data and sessions. Further, encrypting data, more specifically sensitive data, in both transit and storage will provide ample data protection in case of a breach, especially when platform-specific secure storage for sensitive data is implemented. Next, regular updates and patches to software should address vulnerabilities as they are found, or as solutions are presented. Finally, rate limits and firewalls can be utilized to protect against some attacks, like DDoS.