

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 | 03/24/24 | Jennifer Swinton | Update executive summary, Requirements Design Constraints, Domain Model |

**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)

We want to transform the Android-based game, Draw It or Lose it, into a multi-platform, web-based game to improve user experience. The game provides an interactive and exciting challenge where teams compete to determine what is being drawn from displayed images.

In the version of the game we are to develop, multiple teams with multiple players on each team may participate. The team names will be unique, ensuring differentiation among the teams. The software will be built in a way that only one instance of the game can exist in memory at any given time using unique identifiers.

The solution will not only consider the functional objectives of the game but also high-performance. Knowing the game's popularity, the software will be developed to support high use without any affecting the gaming experience.

## Requirements

Business Requirements:

* The game must be web-based, transitioning from the current Android app to multiple platforms.
* The game should allow the participation of one or more teams.
* Each team should be able to have multiple players.
* The game and team names must be unique to avoid confusion.

Technical Requirements:

* Only one instance of the game can exist at any given time. This can be done by creating unique identifiers for each game, team, and/or player.
* The development environment must be set up for the web-based version of this game, as the staff at The Gaming Room does not know how to.

## [Design Constraints](#_2et92p0)

Platform Compatibility: The application should be compatible across different operating systems and web browsers.

Internet Dependence: Being web-based, the game requires an internet connection. Latency could be an issue, especially for real-time games like ours, as the game must be synchronized across different clients.

Scalability: With the increasing popularity of the game, the software application must scale without degrading the game performance.

Data Consistency: Consistent data across different instances of the game is critical for a fair gaming experience.

Security: The game will require players to register and login, leading to data management, and cyber threats could pry on these data.

## [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)

This diagram incorporates five object-oriented classes; ProgramDriver, SingletonTester, GameService, Game, Team, and Player.

The 'ProgramDriver' establishes a relationship with 'SingletonTester' shown by 'uses', indicating that ProgramDriver uses the SingletonTester in its processes.

The 'GameService' class is the class that interacts with Game, Team, and Player as shown by lines labeled '0..\*' indicating an instance of GameService could have zero to many relations with other classes.

The GameService class also observes the Singleton Design pattern (the 'getInstance()'), ensuring only one active instance of the game exists.

The Game class captures multiple Team classes shown by the connection '0..\*', indicating one game instance can have one or more teams. Similarly, one Team can have multiple players as indicated by the '0..\*' .

Last, all classes - Game, Team, and Player point to the 'Entity' class which provides common attributes like 'id' and 'name'.

The interactions and relations depicted in the UML diagram manifest several principles of object-oriented programming such as encapsulation, inheritance, and polymorphism which play key roles in achieving the efficient and sound structure of the software application.

**"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** | Mac offers abilities as a server for hosting web-based applications. It has strong security methods, user-friendly features, and impressive performance. However, its high-cost, proprietary technology, and software compatibility may be problematic. For scalability, it might need additional funds in both hardware and licensing. | Linux is renowned for its high-performance, great security, and compatibility with a wide range of software applications. It's open-source and customizable. However, Linux requires technical knowledge to manage and might not be as good a user experience because of the command-line interfaces. | Windows Server comes with excellent user experience, a lot of compatible software, and available support. On the downside, Windows servers has licensing costs, which might rise with scaling needs. They are also more susceptible to viruses. | While mobile devices can technically host web-based applications, they generally have limited processing power and storage capabilities making them less ideal for large-scale hosting needs. Additionally, security may be a concern. |
| **Client Side** | Mac applications can deliver an excellent user experience, but developing these applications have a greater cost and time due to the need for skilled macOS developers. It may also need specific Apple development tools to build and test applications. | Linux offers a range of possibilities for client-side development due to the open-source tools and libraries. Open-source software implies lower development cost but higher time and expertise because of the learning curve needed. | Windows provides a broad user base, allowing access to larger user numbers. Client application development for Windows would need specialized developers and Microsoft has proprietary tools, which might mean higher costs. | Development on mobile devices requires experience with Android or iOS programming. The large user base means that the greater cost may be worthwhile in user base. Expertise in UI/UX design for smaller screens is required for quality user experience. |
| **Development Tools** | Mac uses tools like Xcode, Xamarin, or AppCode for software development. However, these tools require specific skills and may have additional costs, which increases overhead. | Linux provides a variety of open-source tools like GCC, GDB, Valgrind, etc. While the tools are free, they have a steep learning curve and require knowledgeable developers. | Windows employs tools like Visual Studio, .NET, Xamarin, and many more. These tools are accessible, but they do have licensing costs. | Mobile development requires tools like Android Studio, Flutter, React Native or Xcode(iOS) etc. These tools require specialized programming knowledge and may also have additional costs. |

## 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 utilizing a cross-platform server environment like the Node.js runtime environment. Since “Draw It or Lose It” is trying to be platform-independent, then this would allow the game to run on multiple various operating systems without major changes and allowing high performance and processing.
2. **Operating Systems Architectures**: Node.js is neither a programing language nor a framework, but a cross-platform JavaScript Runtime. This is especially useful for data-intensive, real-time applications running on multiple devices and systems.
3. **Storage Management**: MongoDB can be used as a storage management system for the proposed platform. It is a NoSQL database program that uses JSON-like documents. MongoDB offers high performance, availability, and scalability. This makes it a good choice for managing the real-time, multi-user data the game requires.
4. **Memory Management**: The Node.js platform uses the V8's garbage collector for memory management. Memory management in Node.js is automated, with allocated memory getting deallocated when objects are not used automatically. V8 collects memory into optimized spaces (ie: new-space, old-space, etc.) for quick allocation and deallocation without much delay which decreases lag.
5. **Distributed Systems and Networks**: "Draw It or Lose It" can achieve exceptional communication across multiple platforms using a combination of RESTful API and WebSocket protocol. The RESTful API enables synchronization of data between the server and client during initial loads and updates. In contrast, WebSockets can be used to establish real-time communication between the server and client, transferring real-time game inputs and updates to provide an almost instantaneous gaming experience.
6. **Security**: User data protection and security can be ensured by using encryption during data communication, implementing secure HTTP (HTTPS), and adopting OAuth for user authentication. MongoDB also provides features for data security, including authentication, authorization, and auditing. I'd also recommend adding layers of security in the application to do with user management (like two factor authentication), server protection (like firewalls), and specific application-level protection (like input audit and validation, minimizing attack surface areas, etc). Your data, user application interfaces, and backend servers should be secured and checked regularly.