

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 | 05/16/2024 | Rex Green | Added executive summary and design constraints. |

**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)

The “Draw It or Lose It” application needs to facilitate team-based gameplay, manage rounds and timers, and present randomly chosen images as clues, as well as track guesses and points.

We propose developing a web application to facilitate this game. This software design document will be created for the purpose of outlining the game mechanics and technical specifications. The application will be developed using Eclipse. The final application will require a web server to host the game to be accessible to users, which is outside the scope of this current document. A web-based application will allow users easy access and eliminate the need for on-system installation.

## Requirements

Gameplay requirements:

* Allow for one or more teams per game.
* Allow for each team to have multiple players.
* Guarantee unique team and game names.
* Games consist of four 1-minute rounds.
* Drawings are fully revealed at the 30-second mark.
* The current team gets a full minute to guess. The other teams get a 15 second window to guess if the current team fails to guess correctly.

## [Design Constraints](#_2et92p0)

**Network Latency.** Communication between users and the service can be affected by internet speed and geographical distance. These delays can impact the speed at which users are given data (shown drawings, inputting guesses, receiving round updates) and impact gameplay experience. The application should be designed to minimize data transfer.

**Scalability.** The application needs to be able to handle an unknown number of users playing simultaneously. The server-side architecture should be designed in a way that it can be scaled efficiently to fulfill server demand.

**Single Game Instance.** The requirement to have only one instance of the game in memory at a time. This will be handled in the code using specific data structures.

## [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)

**Classes:**

**Entity.** This is an abstract parent class representing the general concept of a game entity. It has attributes like id (long) and name (String)

**ProgramDriver.** This class is the entry point for the application. It contains the main method which controls the program.

**SingletonTester.** This class is useful for testing.

**GameService.** This class is the central service managing the game logic. It has attributes to the list of games, the game and player IDs.

**Game:** This class represents a game instance. It has attributes for ID, name, and teams.

**Player:** This class represents a player in the game. It has attributes for ID and name.

**Team:** This class represents a team in the game. It has attributes for ID, name, and players.

**Relationships:**

**Association.** Game has a list of Team objects. Team has a list of Player objects. This shows that Game can have multiple Teams and a Team can have multiple players.

**Inheritance.** Player and Team inherit attributes from the Entity class.

**Object-Oriented Programming Principles:**

**Encapsulation.** Data is hidden within classes, being only accessible through methods. This helps ensure data integrity, which is especially important for ensuring that game, team, and player IDs are unique.

**Inheritance.** Code reuse is achieved by inheriting common attributes and methods from a parent class (Entity) to child classes. This promotes code reuse and reduces redundancy, allowing the code to be more easily maintained.

**Aggregation/Composition.** A game can have multiples Teams which can in turn each have multiple Players.

**"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** | - Unix-based operating system, built-in web server  - Reliable performance & robust security features  - Limited market share, not as scalable | - Open source, highly customizable and cost effective, extensive support  - Extremely scalable, large open-source community, secure  - Learning curve | - Widely used server OS, supports various technologies  - Large market share and high familiarity  - Mandatory licensing costs and more resource intensive | - Not suitable for hosting a server |
| **Client Side** | - Maintaining will require developers experienced in Mac, Objective-C, and Swift; experience which might be harder to find and therefore more expensive | - Cost effective  - Offers many development frameworks and open-source libraries  - C/C++, Python, Linux experience. Easy qualifications to find | - Porting Android to Windows likely more expensive as some code will need rewritten  - Porting time can take longer  - Windows developers with C# and .NET experience | - Android is open source (similar to Linux), iOS is Apple proprietary  - Developing a native app for each OS provides an optimal experience  - Developing a native app for each OS is resource heavy |
| **Development Tools** | - Swift is Apple’s modern programming language  - Xcode: Apple’s IDE  - Objective C is an option, but is declining in popularity | - Extensive support for most languages. C/C++, Python, Java, Kotlin, etc.  - Many available IDEs | - C# with the .NET framework, and C++  - Visual Studio, Visual Studio Code, and many third part options  - .NET framework libraries | - Android: Java, Kotlin. Android Studio IDE  - iOS: Swift, Objective-C. Xcode IDE |

## 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**: To allow the Draw It or Lose It application to be easily expanded to other computing environments, the Linux operating system is recommended.
2. **Operating Systems Architectures**: Linux is open source, allowing it to be uniquely customizable and cost effective. Linux is extremely scalable, with extensive community support, open-source libraries and frameworks to take advantage of.
3. **Storage Management** An appropriate storage management system for the Linux server platform is Logical Volume Manager (LVM). LVM is highly scalable; allowing the user to pool storage from multiple physical disks into a single logical volume. This simplifies storage management as The Gaming Room expands and adds more servers. LVM provides additional flexibility by its ability to dynamically resize logical volumes. It can also offer some redundancy benefits. If a physical disk fails, the data on the logical volume can potentially be recovered from the remaining healthy disks in the pool.
4. **Memory Management**: Linux employs various memory management techniques, including virtual memory, paging—the breaking down of processes into smaller units, or pages, which can be swapped out to storage to free up RAM for active processes, and caching.
5. **Distributed Systems and Networks**: The Draw It or Lose It application can achieve cross-patform communication **usin: g** distributed systems and networks by utilizing client-server architecture. The client application (user interface) would run on various platforms while connecting to a central server responsible for game logic, user management, and communication. Network protocols like TCP/IP would ensure reliable data exchange between clients and the server. The server would expose APIs that the client applications on different platforms can use to communicate. These APIs would handle actions like joining games, sending guesses, and receiving game updates. There are some things to consider when using distributed systems. For example, connectively must be maintained between all components of the system. Disconnects can disrupt gameplay for the affected user and potentially impact others if it occurs during critical moments. Network delays can affect responsiveness. Client applications across platforms may need updates to ensure compatibility with server-side changes or bug fixes.
6. **Security:** User data can be protected on and between various platforms via data encryption and user authentication and authorization. Enforce strong password policies for user accounts to prevent brute-force attacks, and implement multi-factor authentication for added security.