

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_heading=h.gjdgxs)

[**Table of Contents 2**](#_heading=h.30j0zll)

[**Document Revision History 2**](#_heading=h.3znysh7)

[**Executive Summary 3**](#_heading=h.2et92p0)

[**Requirements 3**](#_heading=h.tyjcwt)

[**Design Constraints 3**](#_heading=h.1t3h5sf)

[**System Architecture View 3**](#_heading=h.4d34og8)

[**Domain Model 3**](#_heading=h.lnxbz9)

[**Evaluation 4**](#_heading=h.35nkun2)

[**Recommendations 5**](#_heading=h.26in1rg)

## [Document Revision History](#_heading=h.1ksv4uv)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 10/16/2024 | Taylor Colton | Draft of Draw It or Lose It Software Design Document including “Recommendations” section updates. |

**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](#_heading=h.44sinio)

In order to drive more users to Draw It or Lose it, it has been determined that a web-based version of the game is necessary. This design document is intended to facilitate the streamlined development of a web-based version of the game and its environment. The current issues we are facing in regards to the software development process are as follows: unique naming schemes, performance, and the ability to play on numerous devices.

## Requirements

Business Requirements:

The game must be created into a web-based application that can run on numerous platforms, going beyond its existing Android version. It should offer team-based gaming, allowing one or more teams with several players to compete in the game. A unique naming scheme for games, teams, and players is required to prevent duplicates, with users able to check if a name is already in use. To avoid performance difficulties and maintain smooth gameplay, the game must also limit itself to a single instance in memory at all times.

Technical Requirements:  
The game should use the singleton design pattern to ensure that only one instance of the game remains in memory, avoiding conflicts and redundancy. To maintain correct identification and uniqueness throughout the application, each game, team, and player must be assigned a unique identity. The iterator design pattern should be used for methods like addGame() and addTeam() to quickly handle and verify the uniqueness of game and team names. Finally, a base Entity class must be developed to store common attributes like id and name, from which the Game, Team, and Player types will be derived.

## [Design Constraints](#_heading=h.2jxsxqh)

The web-based application will need to cover a few main design constraints: Platform independence, scalability, concurrency, and unique data management. The application should be developed using standard web technologies such as HTML, CSS and JavaScript to ensure it is compatible across numerous platforms. The application should be accessible by multiple users at the same time and able to scale based on user activity. All users within a game should see the same state at any given time in real-time. Finally, teams, users and games should all have unique names that are validated.

## [System Architecture View](#_heading=h.z337ya)

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](#_heading=h.3j2qqm3)

The UML class diagram illustrates fundamental object-oriented ideas such as inheritance, encapsulation, and design patterns. The Entity class acts as a base class, offering shared characteristics and methods for Game, Team, and Player, which promotes code reuse and eliminates repetition. Each class encapsulates its properties and exposes public methods to maintain data integrity and access control. The relationships between the Game, Team, and Player classes are represented as associations, with a game consisting of many teams and each team containing multiple players, resulting in a clear hierarchy.

The GameService class employs the singleton pattern to ensure that just one instance of the service exists at all times, hence meeting the requirement for a single game instance in memory. This class handles game-related actions such as game creation and retrieval, as well as team and player management. The diagram demonstrates the effective use of object-oriented principles to structure the application, ensuring modularity, maintainability, and good management of game components and interactions.

**"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](#_heading=h.1y810tw)

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** | Highest price, yet stable and secure environment for hosting a web app. Not commonly used for enterprise server environments. | Least user-friendly option, but most cost-effective and least resource intensive hosting option. | Expensive licensing and hardware requirements, widely used, and user-friendly GUI. | Not typically used to host applications, yet can host lightweight servers. Severe hardware and security limitations. |
| **Client Side** | Commonly used by users, compatible with most modern browsers, but safari compatibility of the web app should be verified. | Cost effective, but can require significant setup and expertise from users. | Most common OS, so support is critical. Compatible with most modern browsers. | Can involve added development time to support different screen sizes, interactivity, and hardware limitations. |
| **Development Tools** | Commonly used IDEs include Xcode or VS Code. Tools like Docker and Homebrew can be used to manage dependencies and containers. Support for programming languages/frameworks includes Swift, Objective-C and React. | IDEs for Linux include the IntelliJ Suite, VS Code and numerous CLI text editors. CLI tools are essential and widely used. Linux can support development in most modern programming languages. | Visual Studio is a popular IDE available for WIndows that supports development in most programming languages. Also compatible with tools like Docker and WSL. | IDEs used in mobile development include Android Studio and Xcode for developing mobile applications. Development is not typically done on mobile devices. |

## 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**: Linux servers in a web-based, cross-platform approach would be ideal for the web application of Draw It or Lose It. To do this, we would need to utilize web technologies including: HTML5, CSS3 and JavaScript for compatibility across Mac, Linux, Windows, and mobile devices.
2. **Operating Systems Architectures**: Linux (i.e., Ubuntu or CentOS) is ideal for its performance, security and overall support for development resources and web technologies, allowing for efficient management when it comes to web-based game hosting.
3. **Storage Management**: We are recommending cloud-based storage solutions such as Amazon S3 or Google Cloud Storage, along with a relational database (i.e., PostgreSQL) for management of game data, scalability, and reliability.
4. **Memory Management**: Linux, with its inherent efficiency in virtual memory management, paired with a caching solution such as Redis will improve performance and reduce latency for the web app.
5. **Distributed Systems and Networks**: Implement a distributed architecture that uses RESTful APIs or GraphQL for client-server communication and WebSockets for real-time interactions, while guaranteeing system resilience with cloud-based load balancers and CDNs.
6. **Security**: TLS can be implemented for secure data transmission along with OAuth 2.0 for user authentication. These are some base-level security measures we will implement to ensure data security. We will also be utilizing firewalls for enhanced server security, IDS, and using secure coding practices to provide added security when it comes to common vulnerabilities.