

<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 | 11/11/2023 | Braydon Woodward | This is the original documentation for Draw It or Lose It. A team based game, where each team try to guess the picture that is slowly presented to them. |
| 1.1 | 11/26/2023 | Braydon Woodward | In this version the Evaluation section was added. |
| 1.2 | 12/8/2023 | Braydon Woodward | In this version the Recommendations section was added. |

## [Executive Summary](#_sbfa50wo7nsh)

The objective for this documentation is to design a web-based game for our client, The Gaming Room. Draw It or Lose It is a team-based game, where teams compete against each other. Their objective is to guess the picture slowly being presented to them. There can be multiple teams, and multiple players per team. The system should enforce each team and player has a unique name. Additionally, the system should ensure only one game instance exists in memory at any given time.

## Requirements

1. **Web-based game client**
2. **Compatibility with multiple platforms**
3. **Each team has multiple players**
4. **Team and player names must be unique**
5. **Only one instance of the game can exist in memory at a time**

## [Design Constraints](#_2et92p0)

1. **Web-Based Environment:** The game should be accessible through web browsers, so the game can work seamlessly on different platforms.
2. **Unique Game, Team, and Player Identifiers:** The system must create and manage unique identifiers for games, teams, and players. This will require a carefully designed database structure and proper logic to prevent duplicates.
3. **Real-Time Stock Drawing Rendering:** The application will need to render stock drawings to the teams, taking 30 seconds to fully render.
4. **Team and Player Management:** The application should provide a user-friendly interface for players to create a team and add players to the team.
5. **Game Rounds and Timer:** The application should accurately implement game rounds(4) and timers. Each round lasts 1 minute, each rendering takes 30 seconds. If teams do not make a guess before the end of the round, each remaining team gets 15 seconds to offer a guess.
6. **Unique Name Validation:** The system must incorporate checks to ensure unique game, team, and player names during the creation process. Each item will need a specific identifier. Real-time validation and user feedback will be needed.
7. **Singleton Design Pattern:** To enforce only one instance of the game in memory, the program will need to implement the singleton design pattern.
8. **Scalability:** The program should be developed with growth in mind. The system should be scalable to accommodate a growing user-base. Deploying to a cloud base server will ensure the program has enough resources to handle increased traffic.

## [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 UML class diagram shows the relationship between 7 different classes in an object-oriented system. The Entity class is a blueprint for Game, Team, and Player objects that have unique identifiers. It shows the OOP Inheritance, as each of the objects inherits a unique ID and Name within an Entity object. This ensures the design requirement needed for unique Game, Team, and Player objects. The GameService manages Games and their related entities. It follows the OOP Singleton Design pattern ensuring only one instance of GameService exists. It has a 0…\* association with Game meaning the 1 GameService can have 0 or many Game instances. Game and Team have the same association, so one game can have many teams, and one Team can have many Players. This UML shows each object Encapsulates its data, and shows Abstraction by giving public methods to interact with the data. It also has many reused method names like, the toString overrides that show Polymorphism. The UML also shows the ProgramDriver utilizes SingletonTester.

**"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** | Characteristics:  -UNIX based system offers a decent CLI.  -Supports server hosting but is less common.  Advantages:  -Suitable for small to medium scale hosting  -Relatively user-friendly to setup and operate.  Disadvantages:  -Limited server-based software support.  -High hardware and licensing costs for Mac’s proprietary servers. | Characteristics:  -Linux distributions offer strong server deployment capabilities, efficient resource management, and an extensive CLI.  Advantages:  -Open-source software makes Linux servers cost effective, possibly zero licensing fees.  -Linux servers are known to be reliable, stable, and secure.  Disadvantages:  -Requires advanced technical expertise.  -Software compatibility issues, like dependency error, can occur. | Characteristics:  -Servers are designed explicitly for hosting.  -Utilizes a robust GUI.  -Windows has the largest user base of all the OS.  Advantages:  -Good compatibility with commonly used Microsoft technologies, applications.  -Strong support for .NET and Microsoft development tools.  -Easier to use for people familiar with Microsoft ecosystem.  -Higher licensing costs for scaling and multiple connections. | Characteristics:  -Mobile devices interact with the server for authentication, authorization, and data exchange.  Advantages:  -Secure handling of user data and authentication processes centrally on the server-side.  -Consistency in data access and updates help prevent conflicts and data loss.  Disadvantages:  -Increased server load due to requests from multiple devices.  -Potential compatibility issues during server updates. |
| **Client Side** | Characteristics:  -Uses Safari as the default browser.  -Seamless integration with Apple’s ecosystem.  Advantages:  -Consistent user experience within the Apple ecosystem.  -High performance due to optimized hardware-software integrations.  Disadvantages:  -Restricted to Apple hardware for testing and development.  -Limited device diversity compared to the other platforms.  -High licensing fees for development tools. | Characteristics:  -Support for modern web browsers.  -Wide availability of open-source development tools and libraries.  -Works across various hardware configurations.  Advantages:  -Open-source software makes Linux cost effective, possibly zero licensing fees.  -Linux is highly customizable for diverse deployment needs.  Disadvantages:  -Fragmentations across open-source distributions might cause compatibility issues. | Characteristics:  -Support for major browsers like Chrome and Firefox.  -Access to Visual Studio and other Microsoft development tools.  -Works across various hardware configurations.  Advantages:  -Widespread use across desktop environments.  -Access to a broad array of developer tools and libraries.  Disadvantages:  -High licensing fees for development tools.  -Historically susceptible to security vulnerabilities. | Characteristics:  -Various browsers used across different devices.  -Diverse screen sizes and resolutions.  -Varied hardware capabilities can affect performance.  Advantages:  -Mobile devices allow gaming on the go, enhancing portability.  -Enhanced user interaction through touch screens.  -Extensive user-base increasing market reach.  Disadvantages:  -Diverse hardware will result in varying performance.  -Limited processing power and memory compared to PCs or consols.  -Differences in OS versions and device capabilities can cause issues. |
| **Development Tools** | Programming Languages:  -Swift  -Java  IDEs and Tools:  -Xcode 15 (IDE)  -CocoaPods (Dependency Manager)  -Xcode Cloud(Cloud Based Services) | Programming Languages:  -Javascript  - React(Client)  -Node(Server)  IDEs and Tools:  -Visual Studio Code(IDE)  -GCC(Compiler)  -Git(Version Control)  -AWS(Cloud Based Services) | Programming Languages:  -C#  -.NET Framework  IDEs and Tools:  -Visual Studio Code  -Azure(Cloud Based Services)  -Git(Version Control) | Programming Languages:  -Swift(IOS)  -Java(Android)  IDEs and Tools:  -Xcode 15 (IOS IDE)  -Android Studio  -Android SDK(Package Library)  - |

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**: Considering the need for compatibility across multiple platforms and scalability for future growth, I would recommend Amazon Web Services. AWS is one of the top cloud-based platforms that offers all the needed services needed, like computing power, storage, and networking capabilities. This will ensure compatibility between various clients and being able to scale seamlessly.
2. **Operating Systems Architectures**: The OS I would recommend would be Linux. Its open-source nature offers flexibility and customization, as well as cost effectiveness. Linux is well-suited for the cloud environment with support for many different hardware and software platforms while maintaining solid reliability.
3. **Storage Management**: Amazon S3 is an object storage service included in the AWS suite. Storing data in S3 offers a scalable, secure, and reliable storage management solution for Draw It or Lose It.
4. **Memory Management:** When using AWS, memory management will run under the hood freeing resources to be spent elsewhere. AWS uses their proprietary techniques to optimize resource allocation across the game’s software.
5. **Distributed Systems and Networks**: For communication between the various clients and the server, a RESTful API will help ensure seamless interaction. AWS offers an API gateway to deploy and manage the APIs.
6. **Security:** As the leading cloud-service platform, AWS offers all the security features needed. This includes data encryption, access control, identity management and compliance certifications. AWS’s proprietary security features with SSL/TSL/HTTPS network encryption, API authentication, and scheduled security audits will mitigate security threats.