

Draw it or Lose it Web-based App

# **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 | 09/16/24 | Karen Mitchell | Added executive summary outlining the business goals and requirements. Included detailed design constraints and provided rationale for each constraint to show how they align with the client’s goals and budget constraints. Also added Domain model, Evaluation and recommendations. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming room is looking to expand their game “Draw it or Lose it” previously only available for Android, into a web-based game that can run seamlessly across multiple platforms. The game involves teams competing to guess images rendered from a library of stock drawings, the game will have 4 rounds lasting one minute each. To meet the requirements, the solution will focus on cross-platform compatibility, allowing for multiple teams and players, ensuring the uniqueness of team and game names and limiting memory usage to a single instance of a game at a time.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

* *The app must work on several platforms seamlessly and consistently.*
* *The app must be able to have one or more teams.*
* *Teams must have multiple players assigned to them.*
* *Game and team names must be unique, there should be a check to make sure a name hasn’t already been used at the time the name is being chosen and race conditions should be managed.*
* *Only one instance of the game should exist in the memory at any given time.*
* *Drawing render at a steady pace and complete in 30 seconds, If the team guessing doesn’t guess in that time the remaining teams have an opportunity to guess within a 15 second time limit*

## [Design Constraints](#_2et92p0)

1. ***cross-platform development***

*The game must run efficiently across multiple platforms; therefore, a cross-platform development framework should be used. This would allow for a single codebase to be maintained. This would also reduce development time and provide consistency across platforms.*

1. ***Optimization***

*Performance is critical. The game must render images and respond quickly regardless of the device or network. Since there is a large variety of devices and operating systems, The application will need to be optimized to accommodate the different devices such as screen size, processing power and memory.*

1. ***Uniqueness of team/game names***

*For each team and game name to be unique, duplication must be avoided. Implementing the necessary validation and checking methods during name creation will be necessary and race conditions should be managed.*

1. ***Single game instance***

*Only one instance of the game should exist in memory at a time. This would use the singleton design pattern to control instances across the platform. This would ensure no more than one instance of the game is created.*

## [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)

The UML class diagram illustrates the key entities within the "Draw It or Lose It" game and demonstrates the effective use of Object-Oriented Programming (OOP) principles. The **Entity** class serves as a base class that encapsulates shared attributes and behaviors common to the **Game**, **Team**, and **Player** classes. This inheritance hierarchy helps reduce code duplication and promotes reusability by allowing subclasses to inherit and extend common functionality.

The **Game** class is responsible for managing the core game logic, while the **Team** and **Player** classes represent teams and individual players. Multiple players can be assigned to a team, with each player having a unique identifier, ensuring that game participants are unique.

**GameServices** works as the main service manager, handling important tasks like managing the game and team setup, and name uniqueness checks. To ensure that only one instance of the game exists at any time, The Singleton pattern is used in the **GameServices** class, and it's checked and tested by the **SingletonTester** class. The **ProgramDriver** class contains the main() method for executing the game and managing system interactions.

This structure effectively meets the client’s requirements by adhering to OOP principles such as **encapsulation**, **inheritance**, and **singleton design**, resulting in an adaptable, maintainable, and efficient design.

**"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 is stable and secure for hosting web-based apps but may come at a higher cost. MacOS is proprietary and can be expensive compared to open-source alternatives. | Linux is considered a great choice due to stability and cost effectiveness due to it being free and open source and is preferred for hosting high performance apps. | Windows requires purchasing Licensing which can really add to the overall cost, but Windows offers great support for web hosting. | Mobile devices are typically not used for server-side hosting but is a large target platform and compatibility with mobile browsers is important. |
| **Client Side** | Cost of development tools may be higher. Developing for Mac requires considerations of different frameworks and Mac-OS specific features such as swift or Objective-C, potentially increasing training and development costs. | Linux Client side development is less common especially when it comes to end users however its important for developers that may test in a Linux environment so they may require expertise in Linux distributions. | Windows is by far the most common, so compatibility and usability are very important and could reduce cost and time since windows development is generally easier. | Mobile devices are a focus for the client-side development and seamless experience across platforms is important. Investing in a cross-platform framework would streamline development. |
| **Development Tools** | Tools such as Swift UI, Xcode, Visual Code Studio, IntelliJ, flexiHub and CLion are great for building apps allowing for a variety of Languages. Some may be free but specialized tools may have associated costs. | Tools such as Visual Code Studio, git and docker are relevant tools with JavaScript and python being popular languages. Linux has many development tools that are free and open source, reducing costs. | Window development tools include Visual Code Studio, Rad Studio and .NET Project which allow for cross platform development. Windows has both free and paid versions of tools that could impact cost. | For mobile, React Native, Flutter or Unity can be used for cross-platform development. These tools create apps that work seamlessly on both IOS and Android. While some of the tools may be free certain services might add to the cost. |

## Recommendations

1. **Operating Platform**: I suggest using Linux for the server-side because of its cost effectiveness, is scalable, and is known for being secure. For the client-side, using a cross-platform framework like React Native would be recommended as it allows the game to work seamlessly on both Android and iOS, and we can keep the user experience consistent across desktops like Windows and Mac.
2. **Operating Systems Architectures**: Linux is lightweight secure and works well with web-based applications. It’s flexible and helps with managing resources efficiently, which is key in a cloud setup. React Native gives us a modern way to develop mobile apps, letting us share the same code across different platforms.
3. **Storage Management**: A cloud-based service such as AWS or Google cloud would be a good option. These services are reliable and scalable.
4. **Memory Management**: Linux is good at handling memory with techniques like paging and swapping, which comes in handy for big web apps. For the game itself, using the Singleton pattern will help save memory by making sure there’s only one active instance of the game at a time.
5. **Distributed Systems and Networks**: To make sure all the different platforms can talk to each other, a tool like RESTful API can be used. This would let mobile devices, web clients, and servers work together easily. The API could handle things like team registration, syncing game states, and showing puzzles. We should also make sure the network is reliable and has backups, so the game doesn’t crash or go offline.
6. **Security**: To keep everything safe implementing security measures like HTTPS for secure communication and OAuth2 for user authentication, Regular security checks and the Linux built-in security tools would add another layer of protection for the server.