

Draw It or Lose it

# **CS 230 Project Software Design Template**

Version 2.0

## 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 | 07/21/2024 | David A. DiPesa | Our goal is to assist The Gaming Room with the development of their web-based game application called, “Draw It or Lose It.” |
| 2.0 | 08/04/2024 | David A. DiPesa | Updated information in the Evaluation Table Section of this document. |

**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)

This document outlines CTS’s proposed solution to our client, The Gaming Room. They’ve asked for our cooperation in developing a web-based game application called, “Draw It or Lose It.” They channeled inspiration for this game based on the popular, 1980s game show, “Win, Lose or Draw.” The game show had two teams that completed against each other to guess what was being drawn, on a board, by a fellow teammate. Our solution will incorporate a similar idea but, instead of drawing, we will be rendering images chosen from a library of stock drawings, as clues, while multiple teams compete in four rounds of gameplay.

The idea to go web-based with this application is the most logical choice. They want their audience to be able to access it from various types of devices. They also want to provide a unique and fun gaming experience for their users. To meet our client’s needs, we’re going to take into consideration all of their requirements while we deliver an engaging, user-friendly game application that’s truly a blast from the past with a bright future.

## Requirements

The Gaming Room would like to employ CTS’s services for developing a web-based game that utilizes multiple platforms based on their current game “Draw It or Lose It.” The following list of requirements has been compiled so that we can best assist them with this project.

* Our client wants this game to be web-based so that means it needs to be accessible from a variety of different devices, some even using different operating systems.
* The game will support the functional requirements of team participation with the ability to assign multiple players per team.
* The game application will have the ability to enforce a limitation of game instances in memory with the implementation of unique identifiers for the games, teams and players involved.
* Expanding on bullet point three, running sessions of the game and team names should all have unique identifiers to avoid issues with users choosing already taken team names.
* The game has four rounds of play with each round lasting only a minute in duration. The game employs stock drawings to be rendered at a steady rate until they are progressively revealed by the 30-second mark of the round.
* A team’s failure to guess the puzzle before time expires will grant the opposing teams a chance to guess it within a, shorter, 15-second time limit.

## [Design Constraints](#_2et92p0)

In order to make this project successful, I have broken down the project’s design constraints:

* As a technical constraint, the game application must and *will* be developed for a web-based platform.
* As a technical constraint, network, program security and web-browser / device compatibility must all be taken into consideration while we take on this project.
* As a technical constraint, hardware requirements are unknown at this juncture.
* As a technical constraint, only one instance of the game can exist in memory at any given time so we’ll need to focus our efforts with the Singleton Design Pattern.

## [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 following UML class diagram demonstrates several object-oriented programming principles such as inheritance, encapsulation, polymorphism and abstraction. Looking into the diagram, we see the inheritance relationship between the Entity superclass and its three subclasses, Game, Team and Player. Utilizing inheritance enables the subclasses to inherit attributes and behaviors from the superclass (Entity). The three subclasses can utilize the *super* keyword to access Entity's constructor function, reducing redundant code, and ensuring better consistency. Polymorphism is also present in the child classes with the use of toString. The GameService class encapsulates its attributes, such as the constructor and the list of existing games. It only provides methods to interact with the data. This approach maintains data privacy; ensuring only one game instance ever gets created, as well as promotes the principle of abstraction. Abstraction is leveraged by the way the classes manage games, teams and players – concealing unnecessary implementation details and focusing on essential 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](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Unix-based. Very stable and secure foundation for hosting web applications.  A robust ecosystem that’s developer friendly.  Higher cost for hardware / upgrades.  Limited in scalability compared to Linux or Windows.  **Revision 2.0 Updates**: Upon reflection, the GUI is easy to navigate and configuring is very easy to do on the server side. We can run an HTML websocket that will provide a client-side API to connect the browser using a node.js or flask backend server. However, I still believe this is a limited option for us, though it can be done. | A reliable, open-source operating system that allows for customization, flexibility and a wide range of software and tools.  Highly scalable and known for its stability and security in the field.  There are a lot of limitations with the GUI and hardware compatibility.  **Revision 2.0 Updates**: Though Inexpensive it’s just too difficult to navigate. If we have to use it, then Command shell for a simple server configuration is highly likely. | A very wide range of software compatibility and a strong developer ecosystem.  Azure is a good choice since it has flexible growth.  Broad hardware support and documentation.  However, they have the most known security vulnerabilities.  **Revision 2.0 Updates**: This is a frontrunner. It’s got everything we need for a server side solution considering interface and ease of usage but it’s very expensive compared to everything else considered. A saving grace is that we can start small and scale up if needed. | Portability (huge advantage).  Touchscreen, gesture-based interaction or keyboard.  Their screen size can be limited or unappealing to a user.  Varied hardware capabilities (not always a good thing).  Specifications vary from user to user and it’s difficult to optimize depending on current system.  **Revision 2.0 Updates**: We should not consider this option for the server side. I do not see this as reliable and easy to configure. |
| **Client Side** | An intuitive and user-friendly interface.  However, developing and maintaining multiple clients will increase development costs, time and may require a lot of expertise to do.  **Revision 2.0 Updates**: Mac is too expensive and requires too much expertise to use on the client side. Per this evaluation, we *should* no longer consider it.  However, if used, versatile and open-source framework React-native can be implemented to support iOS, Android and Web platform with a single codebase. We can leverage this option and maybe even hire a React-Native developer as a new addition to the team. A more likely solution is to pair that with Linux instead. | Free to use and distribute (important to note).  However, need to consider other costs (hardware and tools).  A steeper learning curve than other systems. May require expertise needed for different clients.  **Revision 2.0 Updates**: Found that the versatile and open-source framework React-native can be implemented to support iOS, Android and Web platform with a single codebase. This is an optimal solution for us.  React Native has the ability to share a single codebase across different platforms, making it platform-independent. I would probably hire a React-Native developer for our team with this solution. Costs will be cheaper. | Licensing costs are always higher compared to the open-source alternatives.  **Revision 2.0 Updates**: Another chance to leverage React-Native here. We can use Visual Studio to accomplish this using JavaScript too. However, here are the other things we’ll have to do: Flask dev, React native web dev, Security dev – authentication and cloud-based solutions.  A bonus is that we can use this with Xbox if they decide to tackle the console world.  Again though, we are always looking at higher expenses with Windows in the end. So even if a good solution, we need to pitch it to The Gaming Room and find out their budgets. | Responsive design and connectivity limitations present.  Familiar user features such as camera, GPS and push notifications.  **Revision 2.0 Updates**: There’s flexibility for clients and developers concerning updates, but they’re more difficult to implement.  We can still utilize React here and be successful with iOS and Android due to responsive media adjustments. Perhaps scaling up for the Web interface is also a possibility too. I do really like the flexibility for updates though and would push for that. |
| **Development Tools** | Node.js and  JavaScript are commonly used with Apple.    IDEs: VSCode and XCode.  **Revision 2.0 Updates**: Development options are good, using Flask, node.js, mySQL, HTML, CSS, and Javascript.  Frontend Development Support: PyCharm, Visual Studio and GitHub. | Robust command-line interface and package management systems like apt or yum.  **Revision 2.0 Updates**: We can leverage Ruby, Firebase, Amazon, AWS and Heroku.  Heroku is nice because it’s a PaaS we can price out based on our terms.  IDEs: VSCode, Atom, SASS and Sublime Text. | C# and .NET framework are popular languages for Windows.  IDEs: Visual Studio, Express.js and even JetBrains.  **Revision 2.0 Updates**: Can use HTML, CSS, React, mySQL and Javascript. Utilize Developer tools such as command prompt, PyCharm and Eclipse. | Kotlin, Swift,  Objective-C,  Java, JavaScript and Android Studio.  **Revision 2.0 Updates**: We need to test across all mobile browsers. This will take time and drive cost up definitely.  We’ll need Javascript so we can leverage iOS and Android application stores for this.  IDEs: XCode / well device emulators and simulators.  Frontend Development Support: Python, HTML, php and C++ |

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**: The Gaming Room wants to expand their audience for Draw it or Lose it. They want to appeal to modern society in the easiest and most robust manner. If we consider these ideas then we want to utilize a web-based operating system. Connecting to the game from any device, be it a smartphone or a desktop PC, is simple. Using a web-based operating system will ensure a broader audience for the game as well as giving a consistent, fun gaming experience for everyone involved.
2. **Operating Systems Architectures**: Our OS architecture is not limited with a web-based operating platform. We’ll want to utilize programming with HTML, CSS and Javascript to play an important role on the client-side of the application. Javascript is actually perfect for this because it will enable us to dynamically update game content. On the server-side, we’ll probably be looking at multi-tier system architecture to handle presentation and application across the board.
3. **Storage Management**: We’ll want to employ a relational database management system (RDBMS). I suggest Oracle since it’s a highly recommended and reliable one to leverage. This coupled with a cloud server / storage system will be the best method to handle data management for the game. RDBMS’s are good for managing structured data like game progress, user profiles and game statistics. The cloud or virtual server, which most companies are switching to now, will be good for storing media and our game’s stock drawing images.
4. **Memory Management**: Memory management can rest in the hands of our web-based operating platform. Web browsers utilize automatic memory management and also have the ability to allocate and de-allocate memory rather than leave it up to developers to always handle this. This will prevent memory leaks and since we don’t know our hardware requirements, it will allow us a lot of flexibility with the application.
5. **Distributed Systems and Networks**: We need to enable communication between various platforms so that the game can utilize distributed software architecture and also leverage network connectivity. Our goal is achievable by implementing a centralized virtual server or cloud infrastructure that serves as the communications hub between game clients. This method is advantageous because a virtual server can handle game launch and synchronization, real-time game updates, as well as messages exchanged between players across different devices. The system should also account for network connectivity issues, which are prevalent with online games. Issues, like intermittent outages or low bandwidth, are avoided by having appropriate error handling and synchronization in place.
6. **Security**: As a reputable game, we’ll want to make sure to protect the users’ personal information. Our chosen web-based operating platform must support secure communication protocols. We’ll setup user authentication and authorization mechanisms, such as a username and password, to control access to the game features and user profiles. Also, data encryption techniques can be used to safeguard sensitive user information stored in databases or transmitted across the network at all times.