

Draw it or loose 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 | 07/18/2025 | Shatoria Giles |  |

**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 Gaming Room wants to turn their Android game, Draw It or Lose It, into a web-based game that works on many devices. The game has teams guessing what is being drawn from a set of images. To make this possible, the software needs to support multiple teams and players, make sure all names are unique, and only allow one game to run at a time in memory. Each game, team, and player will have a special ID to keep everything organized. The team at The Gaming Room doesn’t know how to set this up, so Creative Technology Solutions will help by creating a software design that explains how the game will work online. This plan will show how the parts of the system are connected and meet all the needs listed. Once the plan is approved, we can start building the game for the web.

## Requirements

The Gaming Room wants to make Draw It or Lose It a web-based game that works on many devices. The game needs to support more than one team, and each team must have more than one player. All game and team names must be unique so no one uses the same name. Only one game can run in memory at a time, and each game, team, and player must have a special ID. The Gaming Room also needs help setting up the software so it works online. These are the main business and technical requirements.

## [Design Constraints](#_2et92p0)

Since the game will be web-based, it must work on different devices like phones, tablets, and computers. The internet connection may not always be strong, so the game needs to run smoothly even with slow or spotty connections. Only one game can run in memory at a time, which means we have to manage memory carefully. All game and team names must be unique, so we need a way to check names before they are used. These limits make the game harder to build, but they help it run better for all players.

## [System Architecture View](#_ilbxbyevv6b6)

The system for this game will have different parts that work together. There will be a front end that players see and use, and a back end that handles the game rules, stores data, and keeps track of players and teams. The game will run on a server, and the data will be saved in a database. Players will connect to the game through the internet from different devices. This setup helps the game run smoothly and keeps everything organized. For future projects, we may need to show how all parts connect and how data moves between them.

## [Domain Model](#_8h2ehzxfam4o)

The UML diagram shows how the parts of the game work together. The Entity class is the parent class that gives the id and name to other classes like Game, Team, and Player. This is an example of inheritance, an object-oriented programming (OOP) principle that helps avoid repeating code. The GameService class controls all the games and keeps track of IDs. Each Game can have many Teams, and each Team can have many Players. This shows the composition principle in OOP, where one class is made up of other classes. This design helps the game stay organized and meet the software needs.

**"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 can be used to host web apps, but it’s not very common. It works well with Apple tools and has strong security. However, it’s expensive and not as easy to use for large server setups. | Linux is great for hosting web apps. It’s free, open-source, secure, and runs fast. It’s also the most popular for servers. The only downside is it might be harder to learn at first. | Windows servers are easy to set up and use, especially for people already using Windows. They work well with Microsoft tools, but they cost more and can be less stable than Linux. | Mobile devices are not good for hosting web apps. They don’t have enough power or storage, and battery life is limited. They’re better for using the app, not hosting it. |
| **Client Side** | Mac users expect clean, fast apps. Development costs are higher because fewer people use Macs than Windows. Testing on Mac takes time and special hardware. | Linux is not used as much for everyday users, but some still use it. Apps for Linux are hard to test because there are many versions. It’s cheap and open, but takes time to support. | Windows is used by many people, so it’s important to support. Development is easier and cheaper than Mac. Most developers already have experience with Windows. | Mobile is very important since many users will play on phones. You need to support both iOS and Android, which takes more time and money. Apps must work on different screen sizes. |
| **Development Tools** | Mac supports tools like Xcode, Swift, and Visual Studio Code. It’s great for making iOS apps. You can also run other tools like Node.js, Python, or JavaScript for web apps. | Linux supports many tools like Eclipse, VS Code, and terminal-based compilers. It’s great for Java, Python, and web tools like Node.js. It’s flexible but harder to set up for beginners. | Windows works with tools like Visual Studio, VS Code, and .NET. It’s good for C#, JavaScript, and game engines. It supports a wide range of tools and is beginner-friendly. | Mobile development uses Android Studio for Android and Xcode for iOS. You can also use cross-platform tools like Flutter or React Native. These tools help build apps that work on many 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**: I recommend using Linux as the operating platform for Draw It or Lose It. Linux is free, secure, and works on many devices. It can host web-based apps that players can access from phones, tablets, and computers. Linux is also flexible, so in the future The Gaming Room can expand the game to other platforms without needing a full redesign. Many web servers run on Linux, making it easier to manage and scale the game if the number of players grows.
2. **Operating Systems Architectures**: Linux uses a modular architecture, which separates the core system (kernel) from applications and drivers. The kernel manages resources like memory, processors, and hardware. User programs run on top of the kernel and can interact with it safely. Linux also supports multi-user and multitasking operations, so multiple players can play at the same time without slowing down the server. Its architecture makes it easy to update, secure, and expand the system while keeping all parts of the game organized.
3. **Storage Management**: For storage management, I recommend using a relational database system like MySQL or PostgreSQL. These systems store game data, player info, and team details in tables that are easy to organize and search. They support unique IDs for each game, team, and player. The database can handle multiple users at the same time and ensures data isn’t lost during crashes or power outages. It also works well with Linux, making it simple to integrate into the web-based game.
4. **Memory Management**: Linux uses virtual memory management, which gives each program its own memory space. This prevents one game or player from using all the memory and crashing the system. Linux also handles caching and swapping to keep the game running smoothly, even if many players are connected. Memory is allocated when a new game, team, or player is created and released when they leave the game. This ensures only one game runs in memory at a time and avoids wasting system resources.
5. **Distributed Systems and Networks**: The game can run on a server-client model where the server hosts the game and players connect from different devices. Using the internet, the server communicates with clients through APIs and web sockets for real-time updates. Distributed software allows players on phones, tablets, and computers to play together. Network issues, like slow connections or outages, can be managed by retrying requests and storing temporary game states. The components depend on stable connectivity, so the server must handle multiple connections and keep data synchronized.
6. **Security**: To protect players’ information, Linux supports encryption, firewalls, and secure login systems. Data stored in the database can be encrypted, and communication between the server and clients can use HTTPS. Passwords should be hashed, and unique IDs prevent impersonation. Regular updates prevent hackers from exploiting security holes. Access can be limited so only the server can modify game data. These measures make sure players’ names, scores, and other personal info are safe while using the game on different devices.