

<Name-of-Software-Application>

# **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 4**](#_Toc115077323)

**Domain Model 4**

**5Evaluation 5**

**Recommendations 7**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/21/2024 | Justin Hardin | Complete Executive Summary and Design Constraints |
|  |  |  |  |

**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 game development company Creative Technology Solutions wants a web-based version of their game “Draw It or Lose It”. Doing so will expand the game’s reach by making it available across multiple platforms.

“Draw It or Lose It” is inspired by the TV show “Win, Lose, or Draw”. It is a team-based game that has four one-minute rounds where computer rendered images from a stock library are revealed gradually over time, and the goal is to guess what the image is as soon as possible. If the team whose turn it is fails to guess in time, the opposing teams get an opportunity for one guess within a 15-second window.

Our suggestion for development of the web application is using Java. Using Java makes developing for multiple platforms much easier due to its inherent platform independence. Being a web application, we will be able to reach a broader audience because it will be accessible across multiple devices and operating platforms, increasing active player count and revenue.

## 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.>*

* Must support multiple teams
* Must support multiple players per team
* Each game instance must be unique
* Each team name must be unique
* Only one game instance can be in memory at a time
* Must be accessible to a wide array of devices and operating platforms

## [Design Constraints](#_2et92p0)

<Identify the design constraints for developing the game application in a web-based distributed environment and explain the implications of the design constraints on application development.>

* Must be compatible across multiple web browsers and platforms.  
  - This will limit any platform-specific features to ensure a broader compatibility.
* There can only be on instance of the game in memory at any given time.  
  - Unique id’s should be used for each instance of a game, team, or player.
* Game and team names must be unique.  
  - This will require real-time name availability checking.
* Must support multiple teams and players.  
  - The game should be easily scalable and well optimized to support many players.
* Game state must be managed well to ensure that all clients are experiencing the same thing.  
  - Server to client communication should be efficient and well optimized.

## [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)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

The domain model is represented by the UML class diagram below. It describes the structure and relationships between the classes.

* The Entity class is the base class for Game, Team, and Player. It contains the common attributes shared among the classes, like id and name, as well as shared methods.
* The GameService class manages the instances of the game and creates unique id’s for the game’s entities.
* The Game class inherits from Entity. It contains the list of Teams that are in the game.
* The Team class also inherits from Entity. It contains the list of Players that are in a team.
* The Player class also inherits from Entity. It represents the individual players in the game.

The UML class diagram demonstrates several object-oriented programming principles.

* Inheritance: Game, Team, and Player all inherit from Entity. This allows for the reuse of code for the sake of efficiency.
* Encapsulation: Using public accessors and mutators allows for controlled access to the private attributes.

The composition relationships of Game, Team, and Player make for an organized and efficient creation of a game that has multiple teams made up of multiple players. The GameService class ensures that each game, team, and player has a unique identifier, and it manages each of these. The use of the singleton pattern ensures that only one instance of GameService exists at any given time. This makes managing the game, teams, and players much easier.

**"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)

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** | Has a built-in Apache server. Secure and stable. Higher cost than other options. | Secure, stable, and cost-effective. Great performance. Widely used. | User-friendly. Integrates well with other technologies. Costly | Lack the resources for server side hosting. Better suited for client side interactions. |
| **Client Side** | Supports all major web browsers. | Several browser options. Smallest market share of the main platforms. | Largest market share among desktop and laptop devices. Supports all major web except Safari. | Must be compatible with both iOS and Android. Supports all major web browsers. |
| **Development Tools** | Uses Xcode for native development. Compatible with Java through a Java Virtual Machine. Supports most major cross-platform IDEs | Supports most major IDEs. Has a wide range of customizable development tools. Compatible with Java through a Java Virtual Machine | Supports all major IDEs. Facilitates the use of all major programming languages. Has the widest range of development tools. | Xcode for iOS, Android Studio for Android. Supports cross-platform through frameworks like Apache Cordova, so compatible with Java development. |

## 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 would recommend Windows. Windows is ubiquitous and the most popular platform. It’s user-friendly, secure, and has the widest range of development tools making developing and maintaining the application easier.
2. **Operating Systems Architectures**: We can use Windows Server to host the server side functionality. It has efficient process and thread management, advanced networking features, and flexible system configurations. The Server Kernal will manage hardware resources, process handling, and memory management. It’s well optimized so it should handle the real-time nature of the game with little to no latency.
3. **Storage Management**: We can use separate storage management solutions. For example, storage requirements for game assets like images and audio require different storage solutions than game data like players, teams, and game states. A database would be suitable for storing data, something like SQL Server, whereas for game assets, something like Azure Blob Storage could be suitable.
4. **Memory Management**: Windows has nice built-in memory management capabilities like virtual memory management, dynamic memory allocation for virtual machines. I think it’s important to use a lazy loading technique when loading the game’s images. While it would be nice to be able to load all images into a cache for fast access, I think it’s more important to not overload the memory to accommodate lower-end devices. Since Draw It or Lose It is a web-based game, it will be used by a wide variety of devices, and catering to the more low-end hardware devices ensures that most devices can run the game.
5. **Distributed Systems and Networks**: We could use a RESTful API to handle client-server communication. We’ll use a real-time communication protocol for keeping track of game states and other real-time interactions. For non-real-time operations, we can use ASP.NET Web API
6. **Security**: We can use HTTPS for all client to server communication to keep back and forth traffic secure and have an authentication system in place. I recommend using two-factor authentication when a user logs in. For example, when a user logs in, they have the option of receiving a five digit code either through their email or their phone. Only once they enter in their code will they be granted access to their account.