

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 09/16/23 | Chris Marrs | Created. |
| 1.1 | 09/26/23 | Chris Marrs | Evaluation criteria added. |
| 1.2 | 10/15/23 | Chris Marrs | Recommendation. |

**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)

Draw It or Lose It is loosely similar to the 1980s television game Win, Lose or Draw, where teams compete to guess what is being drawn. Rather than a player drawing images on an easel to help team members guess the puzzle (a phrase, title, or thing), the application will render images from a large library of stock drawings as clues.

The staff at The Gaming Room does not know how to set up the environment. To facilitate the development of the web-based version of the gaming app, we will help in streamlining the development.

## Requirements

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

* Network latency and bandwidth need to be accounted for.
* Server scalability and availability are necessary in any cloud solution.
* Data consistency and synchronization amongst multiple players and games requires real time data management.
* Security and cheating prevention are needed for the protection of data and game integrity.
* Cross-platform compatibility code must support popular platforms while maximizing features of each.
* State persistence and data storage, both for long running games and game interruptions, functionality and data must be state aware.
* User experience and performance need to leverage the functionality of the platforms and overcome limitations of cloud-based solutions.

## [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)

In this data domain model, there are three types of child Entity classes that are inherited from the base Entity class: Game, Team, and Player. A Game can have 0-to-many Teams, and a Team can have 0-to-many Players. This provides an arbitrary combination of games, teams, and players.

As each child Entity class demonstrates object-oriented class inheritance, the extensibility of the base class is also demonstrated as the three child classes, while similar in structure, are different operationally.

The GameService class can service 0-to-many games and their attendant teams and players. The games that can be played are loosely coupled with the teams and players allowing for extensibility and support for different games in the future. The singleton nature of this class is highlighted in the PlayerDriver class as it utilizes the GameService class and demonstrates the singleton via the SingletonTester class.

**"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** | **Cloud Based** |
| --- | --- | --- | --- | --- | --- |
| **Server Side** | No realistic model for server-side hosting. | -Normally NO licensing costs  -Most hosting providers offer Linux servers  -Efficient and stable  -Offers robust server-based deployment    -Open Source | -Extensive support and resources available  -Large software community  -Ongoing licensing costs for server editions  -Server-based deployment  -Proprietary | No realistic model for server-side hosting. | -Vendor specific  -Faster to market  -Correct sized hardware and software  -Scalable hardware and software  -Pay as you go, for what’s used  -Serverless services provide built-in integration. |
| **Client Side** | -Responsive web technologies  -Compatible and optimal performance on all desktop operating systems  -Supports different browsers (Chrome, Firefox, Safari, etc.)  -Provides consistent behavior and appearance. | -Responsive web technologies  -Compatible and optimal performance on all desktop operating systems  -Supports different browsers (Chrome, Firefox, Safari, etc.)  -Provides consistent behavior and appearance. | -Responsive web technologies  -Compatible and optimal performance on all desktop operating systems  -Supports different browsers (Chrome, Firefox, Safari, etc.)  -Provides consistent behavior and appearance. | -Responsive web technologies  -Compatible and optimal performance on all desktop operating systems  -Supports different browsers (Chrome, Firefox, Safari, etc.)  -Provides consistent behavior and appearance. | Any responsive web solution. |
| **Development Tools** | **Dev Tools Applies to all platforms.**  -Front-End  HTML5, CSS3, JavaScript (React.js or Angular.js)  -Back-End  Node.js, Python, or Ruby  -Database  MySQL, PostgreSQL, or MongoDB  -Version Control  Git  Testing and CI/CD  Jenkins or Travis  IDEs  Visual Studio Code (Eclipse requires a lot of desktop environment management across the team)  **Impact on Dev Teams applies to all platforms.**  -Full-stack skills and experience required  -Database design and management required  -Project management required  -CI/CD deployment and testing skills required  -IDE experience will learn or be required  **Cost applies to all platforms.**  Mostly all free or costs based on tiers of usage. | **SAME** | **SAME** | **SAME** | **SAME** |

## 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 recommended platform for The Gaming Room's proposed web-based game system is a backend hosted on Amazon Web Services. This architecture will leverage AWS serverless computing. This backend is designed to work seamlessly across multiple AWS Regions to ensure high availability, low latency, and access to both computing resources and data storage. A failover system will also be implemented that is active-active to improve reliability.

The gaming-client user interface will be built as a responsive web-based Single Page Application using JavaScript and the React JavaScript Library. This will provide a dynamic, interactive gaming experience that is compatible with many devices and screen sizes.

1. **Operating Systems Architectures**:

Authentication – AWS Cognito provides user directory and an authentication server utilizing the authorization service via OAuth 2.0.

Profile, Account, and Marketplace Functions – supported via HTTP RESTful API’s utilizing Lambda serverless functions and NoSQL data storage via DynamoDB.

Single Player Functions – supported via HTTP RESTful API’s utilizing Lambda serverless functions and NoSQL data storage via DynamoDB.

Multi-player Functions – supported via WebSockets utilizing Lambda serverless functions and NoSQL data storage via DynamoDB.

Multi-player Live Scoring Functions - supported via IoT Core and secure Virtual Private Clouds (VPC) utilizing Lambda serverless functions and NoSQL data storage via DynamoDB and real-time scoring data via Redis.

The Client - will be a Single Page Application (SPA) utilizing React that interacts with the backend services via RESTful APIs and WebSockets.

1. **Storage Management**:

There are two storage and memory management approaches on the server side. The use of serverless technology allows for dynamic adjustments to compute and data requirements. The custom services would be written in either C++/C# or Java based mainly on the current staff skill set.

The serverless technology is designed and built to handle storage management behind the scenes and the solution would leverage this. The custom services would rely on some of this inherent management as well, but mostly on good development practices and testing in the release pipeline.

The client side would be dependent on the background memory and storage functions of the browser that the SPA is running in as well as relying on good development practices and testing in the release pipeline.

1. **Memory Management**:

The same as the above in #3 while utilizing garbage collection and customer destructors as needed.

1. **Distributed Systems and Networks**:

Utilizing the AWS cloud architecture allows the backend services to be replicated across Availability Zones and Geographic Regions. If any AZ or GR is down, the applications in the other AZs or GRs take over. With an active-active failover, the routing is automatic, and the real-time data is preserved. The game users should see little to no effects of the failover.

The client would have local cache storage for actions and visual artifacts that would allow seamless operation independent of sporadic internet bottlenecks and any failover that might occur.

The greatest risk presented is if the local internet is down. The user could switch to a cell-based or satellite-based connection and pick up where they were last playing. Any data lost would be minimal. AWS Cognito services provide data synchronization across the different devices that the user might use.

1. **Security**:

The user security functions would be handled by AWS Cognito services. AWS Cognito provides user directory (CIAM) and an authentication server utilizing OAuth 2.0.

Multi-factor authentication (MFA) - provides a level of ‘knowing’ the user above that of the usual userid and password. MFA under Cognito would allow the system to better verify that the user that is logging in, is in fact, the owner of that userid.

Captcha - If there are indications of userid fraud or deceit by an automated attack, Captcha services could be automatically enabled for that userid.

Virtual Private Cloud (VPC) – all multi-user functions that are utilized by game users to directly affect each other are running in a secure VPC that limits access only to services that require their use. No direct user-to-user interaction outside of the VPC would exist and would therefore not be available for exploitation.