

<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 | <01/21/2023> | Brooklen Ashleigh | Initial review of design considerations and recommendation based on client’s needs. |
| 1.1 | <02/05/2023> | Brooklen Ashleigh | Revised Evaluation section of document to better fit criteria. |
| 1.2 | <02/16/2023> | Brooklen Ashleigh | Updated Recommendations section for considerations on how AWS best serves The Gaming Room’s Draw It Or Lose It Project. |

## [Executive Summary](#_sbfa50wo7nsh)

Client The Gaming Room would like to develop a browser-based version of their current Android-only game, Draw It or Lose It. The game will support multiple users in teams playing the same game at once, with the object being to guess the contents of a drawing as it is slowly revealed. The first team to guess before the time limit scores.

## Requirements

* The game will be web-based
* Each game can have multiple teams
* Each team can have multiple players
* Game and team names must be unique
* Only one instance of the game can exist in memory at once

## [Design Constraints](#_2et92p0)

The application will be web-based, meaning it will be accessible by any device using a web browser. As browsers are generally synonymous across operating platforms, the behavior of the application on non-mobile browsers should be consistent. However, the graphical interface must be developed as such to adapt to the constraints of mobile browsers and the performance of mobile phones. Some constraints may need to be put in place for outdated hardware.

## [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 Game, Team, and Player classes all inherit from the Entity class, which provides their unique names and IDs. Because they have private constructors, Games, Teams, and Players are instantiated with unique names and IDs. The Game Service utilizes a Singleton structure that ensures that only one instance of the Game Service can exist at once. Games and Teams utilize unique IDs and names to ensure multiple Games and Teams cannot exist with the same name. Games, Teams, and Players all have a zero to many relation. The Program Driver instantiates all classes,

**"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** | MacOS is not ideal for server-based hosting, as it is locked to specific hardware and is not often used at scale. macOS server has been discontinued. | Linux is excellent for server-side management, as distributions can be customized for the client’s needs and it is excellent for setting up dedicated server farms and scaling. | Windows is also good for server-side management, as it runs on all hardware types and can be used at scale. Microsoft also offers PlayFab, a pay as you go or planned server resource for hosting multiplayer games. Costs depend on compute hours, and a well-supported Premium Plan starts at $2000 a month. | Mobile devices cannot be used to host games at scale, and would therefore require a cloud-based server option or server farm running Windows or Linux. Additionally, cloud-based web services can be purchased from websites like AWS to handle computing needs with costs based on usage. |
| **Client Side** | There are no additional costs or development required for a user with MacOS to run a web-based applications. A single test-bed running macOS will suffice to ensure that the game runs properly in all available browsers. Uniquely, the Safari web browser should be tested, but should not show any significant differences. Proprietary hardware and software must be purchased for this test bed. | Linux installations typically utilize the two most common web browsers, Chromium and Firefox. A single test-bed using a distribution such as Ubuntu can be utilized to test the game’s functionality in these common browsers on Linux. No additional cost, but some expertise may be required. | Windows users typically use Microsoft Edge, Chrome, and Firefox, with some lingering usage of Internet Explorer. Windows requires a licensing fee to install it on a test bed, but it can be installed on any hardware. | Devices running Android and iOS would be necessary to test mobile browser support. Mobile application development for iOS and Android would require no additional cost, but would require additional expertise, as familiarity with Swift and Android SDK are required. There are fees related to serving applications on each platform’s App Store. |
| **Development Tools** | MacOS as a platform supports necessary web-based programming languages like Java, Javascript, and C#. Like other platforms, MacOS has access to popular Java IDEs like Eclipse, NetBeans, intelliJ, etc. | Linux distributions as platforms support necessary web-based programming languages like Java, Javascript, and C#. Like other platforms, Linux has access to popular Java IDEs like Eclipse, NetBeans, intelliJ, etc. | Windows as a platform supports necessary web-based programming languages like Java, Javascript, and C#. Like other platforms, Windows has access to popular Java IDEs like Eclipse, NetBeans, intelliJ, etc. | Mobile apps can be built on Windows, macOS, and Linux platforms. iPhone applications are generally made with Swift in Xcode, and Android apps with Java in the Android SDK. These development methods do not have fees, but require expertise. |

## 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**: As Draw it Or Lose It will be non-graphics intensive, browser-based game that requires multiplayer support and web connectivity, we recommend that The Gaming Room uses Amazon’s Web Services as an operating platform to both store the game’s assets and run its back-end server requirements. Amazon Web Services, or AWS, is a cloud-based web architecture that serves web applications with a host of various tools needed to host complex and scaled web applications, and would therefore be perfect for the Draw It or Lose It project.
2. **Operating Systems Architectures**: AWS’s Amplify service handles authentication, data storage, analytics, and serves server-rendered web applications, thereby making it perfect for the Draw It Or Lose It application. With this client-server relationship, AWS would handle all server-side needs, including user authentication and management, as well as file storage and the very important serving of images that game players need to guess. The client side would be a simple web interface based in HTML and Javascript that would render the interface of the application and communicate with AWS.

AWS Device Farm allows the testing of web applications on multiple types of devices and form factors, allowing The Gaming Room’s development team testers to ensure quality across all types of platforms: Desktop browsers, mobile browsers, tablet browsers, etc.

Pricing for AWS is calculated by the processing minutes needed to carry out the server side of the application, and therefore by how much demand the application requires multiplied by how many users it serves.

1. **Storage Management**: Amazon Amplify uses Amazon’s S3 service to store data and serve the images necessary to play the game. Draw it Or Lose It will pull resources from remote directories as needed when the game is played, then discard them when each individual game is finished. This will ensure that the app does not require a large number of assets to be downloaded and stored on the user’s device, and individual images can be served easily to the user without noticeable delay. This method will also help prevent the mining of images, as the client device will only have access to one image at a time.
2. **Memory Management**: The server side of the game uses a singleton method to ensure that only one instance of the gameservice is in memory at once. Then, user and game objects are created as needed as various client applications add users and game instances. These objects would need to be dumped on both the client and server side every time a game ends, with only the game the user is actively playing in memory as well. Amazon Amplify has the ability to cache assets to optimize application performance as needed. It also allows fine-grained control of memory, including adding soft limits to particular tasks for the sake of optimization.
3. **Distributed Systems and Networks**: As this game will be web-based, all assets can be served with a distributed client such as Amazon Web Services as mentioned above. All modern browsers, independent of operating system, will be able to fetch and interact with this server contingent on a network connection. The downside of this is requiring a connection to be able to connect and play with other people, but this is a common limitation. Using a service like Amazon’s Amplify or EC2 would mean low downtime and the added advantage of not having to maintain dedicated servers. Due to AWS being a cloud-based service, downtime is possible, but outages are rare (a few times a year) and typically do not last longer than a day. Not having the need to keep staff and space needed to maintain those servers is also mitigated, and relying on a service like AWS offers much less risk and far more reliability.
4. **Security**: Amazon Web Services supports server-side encryption, which will mean that communications between the web servers and devices accessing the game will be secure and encrypted. Web Services offer a host of encryption types and key encryption through AWS CloudHSM. The game should not store information about its users locally, and would use Amazon’s DynamoDB database service. This database service is encrypted, and also offers backups and secure restoration of lost data, ensuring that if something goes wrong, service can be restored to its proper state as needed.