

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[CS 230 Project Software Design Template 1](#_Toc673335368)

[Table of Contents 1](#_Toc1682754119)

[Document Revision History 2](#_Toc1658607635)

[Executive Summary 2](#_Toc2028876260)

[Design Constraints 3](#_Toc439814455)

[System Architecture View 3](#_Toc650967570)

[Domain Model 4](#_Toc1087234566)

[Evaluation 5](#_Toc72439877)

[Recommendations 5](#_Toc2052143737)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 04/15/22 | Adam E. Garcia | V1.0 Initial Summary and Design Layout + Server-Client-side recommendations + Architecture Recommendations |

**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 develop a web-based game that can run on multiple platforms. The game will be called “Draw It or Lose It” and is currently only available on Android as an app. The purpose of the game is for several teams consisting of multiple players competing against each other in four rounds lasting a minute each. A drawing is rendered from a library of stock images, and one team attempts to guess the image before time runs out. If the team fails to correctly guess the image, each opposing team member gets an opportunity to answer until time runs out.

The client requires one or more teams to be involved in play with multiple users assigned to each team. The game and team names must be unique, and a solution must be added to check whether a name is in use before choosing a team name. Also, only one instance of the game is allowed to exist in memory at any time, and a solution must be added to check for unique game identifiers to ensure only one instance exists at any time.

## [Design Constraints](#_2et92p0)

* One or more teams involved.
* Each team can have multiple players assigned.
* Game and Team names must be unique to allow users to check whether the name is available.
* Only one instance of the game can exist at any time.
* Must run on multiple platforms

These are the requirements needed to follow when developing the application. The client requires the application to run on multiple platforms, including Android, iOS, Windows, macOS, and Linux. We can achieve this by developing the application in a web-based environment using universal coding languages that can be interpreted on various platforms and browsers.

## [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 Entity class creates a relationship between Game, Team, and Player class. These classes inherit attributes and methods from the Entity class, such as “name” and “id”. This makes Entity a superclass. This is a demonstration of aggregation, as each subclass has an instance that references an instance in another class. For example, the Teams class references instances of Players, the Game class references instances of Teams, the GameService class references an instance of Game, etc. We can see these relationships demonstrated in the UML diagram below.

**"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** | Powerful terminal support.  Popular in web hosting.  Upgradable. Diverse options for different web hosting requirements.  Less support for web hosting than other platforms. | Powerful terminal support.  Secured, most preferred for webhosting.  Strong community support provides quick updates and patching when flaws are exposed.  More difficult for third-party application support. | Most third-party application support compared to other platforms.  Maintains market dominance. High resource requirements, less loading time, most user-friendly.  Susceptible to most security flaws. | Hard to maintain a static network address on a mobile device for webhosting.  High portability.  Large reach. Cost-effective.  Poor security. Lower resources. |
| **Client-Side** | Moderate expertise and time required. The cost is similar to Windows. | Maximum expertise and time required. Minimum cost. | Minimum expertise and time required. The cost is similar to macOS. | Provides flexibility to clients or even developers to see updates at any place. Slightly more difficult to implement than other devices. |
| **Development Tools** | IDEs and tools include VS Code, Eclipse, Atom, Notepad++, etc. Languages include HTML5/CSS/Java/Java/C/C++/Python/Swift/PHP/Ruby. | IDEs and tools include VS Code, Eclipse, Atom, Sublime, Notepad++, etc. Languages include HTML5/CSS/Java/Java/C/C++/Python/PHP/Ruby. | IDEs and tools include Visual Studio, VS Code, Eclipse, Atom, Notepad++, etc. Languages include HTML5/CSS/Java/Java/C/C++/Python/PHP/Ruby. | IDEs and tools include Visual Studio, VS Code, Eclipse, CppDroid, AIDE, etc. Languages include HTML5/CSS/Java/Java/C/C++/Python/Swift/PHP/Ruby. |

## Recommendations

1. **Operating Platform:** To meet the needs of the client's needs for system flexibility and maintain low upfront costs to host the application, a cloud-based, serverless environment such as Microsoft Azure would be recommended. This allows the client to host the application with a minimal upfront cost and then expand their resources of their hosted cloud system as they make updates to the application and the application reaches more users.
2. **Operating Systems Architectures:** This flexibility of a serverless architecture such as Azure allows for function extensions in IDEs such as Visual Studio and VS Code to be fully integrated with the Azure platform. The architecture also grants the client the ability to choose between hosting the application on a Linux or Windows based code-only OS with a variety of coding languages available to integrate into the application, such as C#, Java, JavaScript, Python, etc., as well as API’s like .NET.
3. **Storage Management:** Azure has several options for managing storage including Archived Storage for low-cost, durable, and highly available secure cloud storage for rarely accessed data, Blob Storage for massive scalability for unstructured data, and Azure Files for simple, secure, and fully managed cloud file sharing storage. These solutions are highly flexible and cost-effective, with costs as low as $0.002/GB per month.
4. **Memory Management:** Azure can pull the stock images stored on the service, allowing for low memory overhead. Memory management should not be an issue since only one game will exist in memory at any given time. Again, we can have an initially low upfront cost for resources and scale memory allocation further as the application expands and has further user reach.
5. **Distributed Systems and Networks:** Using a cross-platform IDE is key to developing an application that is compatible with universal languages and dependencies. Once the application is developed and hosted on the web, cross-platform should not be an issue as the game will be able to run on any chromium-based web browser. Azure offers a Virtual Network option that connects to the web as well as with on-premise resources and even with other Azure instances. This option is highly scalable as the application continues to grow and has more reach. The one downside of the Azure network environment is that these functions cannot be directly accessed through traditional means such as their IP addresses and instead need to be integrated with private APIs for this sort of accessibility.
6. **Security:** Microsoft Defender for Cloud is a strong solution to protect from most system attacks. Defender updates its definitions regularly when new threats have surfaced. A database for player login names and passwords should be established with standard encryption protocols to avoid user information being obtained in the event of a potential breach. This can be achieved with a cloud container, such as Azure SQL Database.