

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc142842582)

[Table of Contents 2](#_Toc142842583)

[Document Revision History 2](#_Toc142842584)

[Executive Summary 3](#_Toc142842585)

[Requirements 3](#_Toc142842586)

[Design Constraints 3](#_Toc142842587)

[Domain Model 3](#_Toc142842588)

[Evaluation 4](#_Toc142842589)

[Recommendations 5](#_Toc142842590)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/13/23 | Isaac Ocegueda | Update cover page, executive summary, requirements, design constraints, and domain model. |
| 1.1 | 07/27/23 | Isaac Ocegueda | Update the Evaluation section. |
| 1.2 | 08/12/23 | Isaac Ocegueda | Update the Recommendations section. |

**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 would like to make their web-based game, Draw It or Lose It, available as a web-based game on multiple platforms. This game is currently only on the Android platform. To solve this problem, we will create an environment using the Java coding language that executes all the game’s software requirements as listed below in this document.

## Requirements

Business requirements:

* Make the game available to users outside of the Android platform.

Software requirements:

* Game can have one or more teams.
* Each team can have multiple players.
* Checks for game and team name uniqueness.
* Only one instance of the game exists in memory at any time.

## [Design Constraints](#_2et92p0)

1. Code a Java program that adapts the Android game into a web-based game:  
   The program must follow the same rules of play as the current Android game. This includes having teams compete against each other to guess from a computer-rendered drawing within a certain time limit (30 seconds). If the first team cannot guess the image, the other team has one opportunity to steal the point by guessing within the time limit (15 seconds). This pattern repeats for 4 rounds.
2. The program must only have one instance of the game happening at a time:  
   This requirement ensures that the web-based game runs well and the team points are accounted for correctly. It also helps with computer memory issues that could arise if many games are started, either on purpose or by accident, at the same time. Using a singleton programming approach can help solve this design constraint.
3. The program checks for game and team name uniqueness:  
   This requirement ensures teams have different names and games to be named/numbered correctly for the users. Creating unique identifiers for each instance of a game, team, and player will solve this design constraint.

## [Domain Model](#_8h2ehzxfam4o)

The design model below shows the different package relationships within the system. The Entity Class is designed as a parent class to the Game, Team, and Player classes. These three subclasses inherit identifier attributes and methods from the Entity class. These three classes also have a multiplicity relationship with each other, as in there can be zero to many players in a single team and zero to many teams in a single game. Once all the players and teams are added to the Game class, the GamerService class takes over. The Game and GameService class also have a multiplicity relationship where there can be zero to many games used by GameService. The ProgramDriver class is the main class and uses the SingletonTester to ensure that there is only one instance of the GameService class at a time.

**"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** | Pros:   * Large database and support for creators * Strong security measures * Good GUI   Cons:   * Closed ecosystem with specific hardware * Most expensive option. | Pros:   * Free to use * Open source * Lots of development tools   Cons:   * Requires more coding knowledge | Pros:   * Most support of application and third-party software * Lots of development tools * Uses common coding languages   Cons:   * Licensing can be expensive * Not as secure | Pros:   * Good developer tools * Not as expensive   Cons:   * Require constant connection to internet * Requires specific coding knowledge |
| **Client Side** | Pros:   * Support for multiple browsers and tools * Lower development time   Cons:   * Requires Apple products that run MacOS | Pros:   * Open-source means development for every web browser * Many support tools   Cons:   * Development time based on coding experience | Pros:   * Many support tools * Quick development time * Quick deployment   Cons:   * Security measures must be designed * Cost of some development tools | Pros:   * Portability and access * Development tools accessible   Cons:   * Longer development time * Hard to test from mobile to other environments |
| **Development Tools** | * Swift * Objective-C * Xcode IDE | * C programming languages * Python * Visual Studio * Eclipse IDE * IntelliJ IDEA IDE | * C programming languages * .NET * Visual Studio * Eclipse IDE | iOS:   * Xcode for iOS   Android:   * Java * C programming languages * Python * Eclipse IDE * Virtual Studio |

## 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**: While it might seem like Windows or MacOS are great operating platforms to use for the game’s development and operation, the operating platform that offers the most useful features and control is Linux. Linux is a free and open-source platform that gives developers control over code modification and maintenance, i.e., fixing bugs. Linux also has a wide range of support from other developers to help ensure the game keeps working in case of technical problems arising. Linux also offers excellent privacy and security features that are built into the system and do need to be purchased like in other platforms.
2. **Operating Systems Architectures**: For the purpose of allowing the game to run on multiple platforms, including Windows, MacOS, Android, and iOS, a web-based application is the best option. A web-based, or multi-tier, architecture allows for the game’s storage, memory, and the player’s viewing device to be separated into different parts and hardware/software. For this game, the player will access the game through a web-browser, meaning that any device with an internet connection can access it, and then the game will be managed through a cloud storage and memory manager (refer to storage and memory management sections). This multi-tier architecture with cloud components allow for a cheaper way of running the game as it prevents the company from having to purchase and maintain physical servers.
3. **Storage Management**: Given that the game will be web-based, cloud storage is the best option to use. A good source of cloud storage is Amazon’s AWS cloud solutions, which only charges for the space used. This is a good option since the picture library for the game might start out small but can grow to many thousands of high-definition pictures. To put a price on it, Amazon currently offers storage of up to the first 50 TB for $0.023 per GB. Even if there were 100,000 HD pictures, the price on that would be $18.40 a month. This is a great price if we consider how much it would be to buy physical servers that then need to be set up and maintained.   
     
   AWS is also a Linux-based server. This itself has many benefits, such as reliability, flexibility, and scalability. It is with good reason that many big corporations like Google use Linux-based servers. They have great tools and resources for growth and maintenance. They also do not need to be restarted, unlike Windows and MacOS platforms, when changes are made, making them very reliable. Lastly, the open-source aspects makes these servers cheaper than other platforms, hence the great price on the AWS server for storage.
4. **Memory Management**: Quickly transferring image data to all the players is one of the most demanding processes. Each image must be transferred from the server to the player’s device and rendered within the 30 seconds between each round. Add to this the complexity of each player possibly having different devices, from smartphones to gaming computers, and figuring out a reliable way to manage and transfer data into device memory becomes a priority. Using a serverless architecture to manage the game’s memory is the best option, especially since storage will be cloud-based as well.   
     
   In a serverless architecture, the cloud-based server quickly processes the request for each picture and sends it to the client’s application where it is loaded into cache memory and rendered client-side. Cache memory allows for quick access to the picture’s data and is what allows each picture to be loaded within the 30 second time limit. After the picture has been fully drawn and guessed, the picture is deleted from cache memory and the new picture is loaded in. This process repeats until the game ends.
5. **Distributed Systems and Networks**: Connectivity between the local Linux server, cloud server, and player gaming device must be secure and quick. For this reason, using a RESTful API is the recommended setup. REST has several benefits to the speed and security of the game. This section will go over some of the speed and communication benefits, and the next section will go over the security benefits.  
     
   In a RESTful API, the multi-tier architecture makes and processes requests through HTTP. This means that the game can be processed through any web browser regardless of platform. The data sent, in this case player, team, and picture data, are cacheable data making it streamless for client-server interactions. Since the system is stateless, it removes server load and promotes scalability without reducing server performance.
6. **Security**: Linux on its own as a platform is very secure. Even though it is open source and free, the amount of people looking at the code and patching/debugging it makes it so that any exploits to the platform are quickly removed. Since we are using a Linux-based cloud server as well, these benefits apply to it as well. On top of the security that Linux offers, using a RESTful API architecture offers even more privacy and security benefits.  
     
   One of these benefits is in the stateless aspect of REST. Since each communication request is separate and not connected to any other requests, no client information is stored between get requests and therefore even if communication was intercepted, it would not contain any player’s personal information.  
     
   Another benefit is the integrated JSON encryption in RESTful APIs. JSON encryption server architecture uses authentication and authorization by keeping all the information restricted to the server-side. This means that for user data to be breached, it must first go through the hurdle of hacking into the Linux-based server. This is not an easy task and ensures that players’ personal data is always safe and secure.