

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0.2

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0.0 | 03/17/2022 | Austin Frey | Completed Executive Summary, Design Constraints, and Domain Model sections |
| 1.0.1 | 04/03/2022 | Austin Frey | Completed Development Requirements section including Server Side, Client Side, and Development Tools |
| 1.02 | 04/07/2022 | Austin Frey | Completed Recommendations including Operating Platform, Operating Systems Architectures, Storage Management, Memory Management, Distributed Systems and Networks, and Security |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room seeks to expand upon their game application that is currently only available on Android. Creative Technology Solutions (CTS) will assist The Gaming Room with streamlining the development of the web-based version of the game. To create a web variant of Draw It or Lose It, The Gaming Room will have to set up a new hosting environment for the application. Specifically, they will need to acquire a domain name, a domain host, and a client-server for the game to operate. CTS will not provide the funding for acquiring the previous requirements, but CTS can acquire the resources on behalf of The Gaming Room if so desired. Once CTS is provided (or acquires) the necessary resources, CTS will begin developing the environment for the web-based version of the game.

The Gaming Room has requested that the following software requirements are considered when creating the environment for Draw It or Lose It:

* 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

CTS will ensure that the requirements above are met with the development of the web-based environment before releasing the project back to The Gaming Room. The "Design Constraints" section of this document will elaborate on the specifications of meeting the software requirements above.

## [Design Constraints](#_2et92p0)

**Business Constraints**:

* The Gaming Room holds all rights to the final product.
* CTS will only be responsible for developing the environment for Draw It or Lose It to operate on. CTS will not be responsible for creating game functionality.

**Technical Constraints**:

* Draw It or Lose It should be platform independent, thus the environment will be created using the Java programming language.
* A game will have the ability to have one or more teams involved, so an object-oriented design will be implemented.
* Each team will have multiple players assigned to it, which should be checked before starting a game.
* Game and team names must be unique. This means that no game or team name should repeat across *any game instances.* At creation time, all game and team names will refer to a global list to ensure compatibility.
* Only one instance of a game can exist at any given time. This will require unique identifiers for each instance of a game, team, or player, as well as a deep comparison between game objects to ensure the same game does not exist twice.

## [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)

**Class Explanations:**

* GameService: This is the class that will act as a global moderator for all class instance comparisons to ensure only one instance of any player, team, or game exists
* Game: This class is what will hold the actual game. It also includes a list of Teams associated with that instance of a game.
* Team: This class will have a list of players.
* Player: The "smallest" piece of the design. The Player will only have the information contained in entity.
* Entity: The Entity class is exists for the sake of minimizing redundancy when creating Player, Team, or Game instances, which is why those three classes inherit from Entity.

**Class interactions:**

* GameService – Game: An instance of GameService will only directly interact with the Game class, and will hold zero or more instances of Game within itself.
* Game – Team: Game will directly interact with Team and will hold zero or more instances of Team within itself. Game will also directly interact with GameService, but will not have any direct interactions with Player. It also inherits attributes from the Entity class (id and name).
* Team – Player: Team will directly interact with Player and will hold zero or more instances of Player within itself. Team will also directly interact with Game. Team will inherit id and name attributes from the Entity class.
* Entity – Player, Team, Game: Entity provides a method of storing attributes for the object classes, excluding GameService. Entity could be created as an abstract class, but it was decided that Entity will be treated as a base class, rather than an abstract class.
* ProgramDriver – SingletonTester: ProgramDriver uses SingletonTester to ensure that only one instance of GameService exists at any given time. Not doing so could result in program failures.

**Object-Oriented Principles:**

* *Abstraction*: Not currently applicable to this project.
* *Encapsulation*: All attributes are hidden (a.k.a. private). Getters are utilized for accessing data, but there are not setter methods. This helps ensure unique names because it only allows assigning a name upon declaration of the object.
* *Inheritance*: Game, Team, and Player all use inheritance from Entity to enable id and name accessor methods, as well as ensure that Game, Team, and Player all have an id and name attribute.
* *Polymorphism*: Polymorphism is not taken advantage of in the current implementation of this project. However, the inheritance design enabled by the Entity class lays the groundwork for Polymorphism. One example of how polymorphism could be utilized is a delete function, which could accept a game, team, or player, and delete all data associated with that object.

**"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)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Server side | | | | |
| Dev Requirements | Mac | Linux | Windows | Mobile Devices |
| ADP Support | Yes | Yes | Yes | No |
| Availability | Proprietary | Open source | Proprietary | Varies |
| Cloud support | Yes | Yes | Yes | No |
| Customer Support | Excellent | Varies | Good | Low |
| Hardware Requirements | High-end (Proprietary) | Low-end | High-end | Minimal |
| Hardware Base Price | $999 | $75 | $299 | $100 |
| LDAP Support | Yes | Yes | Yes | No |
| Licensing Cost | $499 (10 clients)  $999 (Unlimited) | Free | $501 (Essentials[1])  $1,069 (Standard[2])  $6,155 (Datacenter[3]) | Varies  Min: Free  Max: Unknown |
| Offers Server-based Deployment | Yes | Yes | Yes | Yes |
| Operation Method | GUI | Terminal | GUI | GUI |
| Scalability | High | High | High | Low |
| Security | High | Medium | Low | Low |
| Stability | High | High | Medium | Low |
| Updates | Purchase | Free | Purchase | Varies |
| Unique Characteristics | * OS X server requires Apple hardware * Apple offers an "update plan" for an additional charge * Client limit is solely for file-sharing | * Can be installed on any PC * Terminal operation can be difficult for users with no technical experience | * Most commonly used operating system in the world makes operation easy for most users * [1], [2], and [3]: see [Windows Server](https://www.microsoft.com/en-us/windows-server/pricing) website for details | * Any android device can serve as a small webhost server * Several apps available to convert android phone into a server |
| Ideal for: | Large scale operations | Any size operation | Small to Medium size operations | Small, static websites with minimal visiters |
| Advantages | Reliable, secure, scalable, powerful, intuitive GUI, Great customer support | Free, open source, flexible, reliable, powerful, semi-secure, scalable | Familiar GUI, medium-budget, scalable | Low cost, GUI, minimizes hosting costs |
| Disadvantages | Expensive, pay for updates, requires very specific hardware | Technical prowess required, limited customer support, learning curve | Insecure, unstable, somewhat expensive | Insecure, weak, limited functionality, short list of services |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Client Side | | | | | | | |
| Dev Requirements | Mac | | Linux | | Windows | Mobile Devices | |
| Cost | Standard | | Additional employee costs for developing in addition to MacOS and Windows | | Standard | Additional costs for employees' work and development tools if the team desires to create the app outside of the desktop browser variant. | |
| Time | Standard | | Requires additional time to create a comprehensive application that works with all instance of Linux OS | | Standard | Additional time required to modify the game to fit mobile screen dimensions and interface with touch screens | |
| Expertise | May require additional expertise or training if individuals are unfamiliar with MacOS | | Additional proficiencies required since Linux has so many variations | | Standard | Additional proficencies required for creating mobile applications | |
| Solution | The client side requirements consideration is generated on the assumption that most developers are familiar with Windows since it is currently the most common operating system. Regardless, all issues presented above can all be solved by developing with a cross-platform SDK such as [Corona](https://coronalabs.com/), [Unity](https://unity.com/), [Cocos2d](http://cocos2d.org/), or another SDK of the development team's choice. However, the drawback of using a cross-platform SDK is that developers are unable to use new OS features until the maintainer of the SDK has updated the SDK to interact with the new OS. One other aspect worth noting is that the client-side requirements assume the application will be running on the desktop. However, since the application will be running through a browser for Mac, Linux, and Windows platforms, system compatibility is not as much of a concern as browser compatibility. Cross compatibility should be tested rigorously through the application development process. | | | | | | |
| **Development Tools** | | | | | | | | |
| Dev Requirements | | Mac | Linux | Windows | | | Mobile Devices | |
| Supported Programming Languages (includes but not limited to) | | * C * C++ * **CSS** * **HTML** * **Java** * Python * Swift | * C * C++ * **CSS** * **HTML** * **Java** * Python | * .NET * C * C# * C++ * **CSS** * **HTML** * **Java** * Python | | | * C * C++ * **CSS** * **HTML** * **Java** * JavaScript * Kotlin * Python * Swift | |
| IDEs | | * Eclipse * PyCharm * **Visual Studio** (May require purchasing a license) * Xcode | * Eclipse * PyCharm * **Visual Studio** (May require purchasing a license) | * Eclipse * PyCharm * **Visual Studio** (May require purchasing a license) | | | * Android Studio * Eclipse * IntelliJ * **Visual Studio** (May require purchasing a license) | |
| Tools | | * **Java Development Kit (JDK)** * **Java Runtime Environment (JRE)** * NodeJS * Python | * **Java Development Kit (JDK)** * **Java Runtime Environment (JRE)** * NodeJS * Python | * **Java Development Kit (JDK)** * **Java Runtime Environment (JRE)** * NodeJS * Python | | | * **Java Development Kit (JDK)** * **Java Runtime Environment (JRE)** * NodeJS * Python | |
| Solutions | | The bold-font sections above highlight the options that would be optimal for developing a cross-platform application. Java, as stated earlier, is the ideal language for development, and that includes the client-side. The client side will also need to utilize CSS and HTML for display purposes. It is best practice to have developers all working in the same IDE for the sake of consistency when sharing files. Eclipse and Visual Studio both could suffice for this purpose. Eclipse is a free software, however, Visual Studio offers more flexibility and scalability if the organization purchases a license. If one team of less than five members is developing the front end, Eclipse will suffice. If more than five members, or multiple teams, will be developing the client side, then Visual Studio will likely be more beneficial. Lastly, the JDK and JRE tools must be installed on developers' and the users' systems to test and run the application. | | | | | | |

## Recommendations

1. **Operating Platform: Linux**

The ideal choice for the server's operating platform is Linux because of its flexibility, scalability, stability, security, and cost-efficiency. There is a slight learning curve for individuals not familiar with Linux, but the cost of training or finding someone familiar with Linux is negligible compared to the expense of operating a server using macOS or Windows.

1. **Operating Systems Architectures: Web-based Application (Multi-tier)**

Web-based application architecture is the best choice for the Draw It or Lose It game because it is designed to work with browsers, which is the primary client The Gaming Room intends the game to be played on. This also eliminates the need to develop for several platforms, which drastically reduces the overhead of creating the application. There are three tiers typically in a web-based application: the server, the middle tier, and the client. Linux is the best operating platform for the server and middle tier for the same reasons as above, but that still leaves the question about the client's operating platform. The client's operating platform is negligible since the game will be played in a browser. However, developing for multiple browsers should be considered during development. In most cases, there are no significant differences between programming for the various popular browsers (Mozilla Firefox, Google Chrome, Safari, Opera, Microsoft Edge, etc.). However, new or experimental features of JavaScript, HTML, or CSS should be avoided because not all browsers support the latest features.

1. **Storage Management**: **There is not enough information to answer this prompt**

*Missing Information:*

* Are users signing up for the application to play, or do they simply generate a name? If they sign up, the server system must now consider storage space for user information. Additionally, the file system storage structure will be different if the information on the server is dynamic rather than static.
* Is the library of images expected to increase for Draw It or Lose It? This is an essential consideration for data storage space requirements.
* What are the current hardware requirements for the application's operation? Since the server will be acting as a web server and a middle-tier, the game's current hardware requirements need to be considered for determining hardware requirements for the server, such as CPU, RAM, and Storage.
* Is the game expected to continue to be developed in the future? For example, are there plans to expand the game or the server's functionality?

We will make a few assumptions about the answers to the above questions to answer this question.

* Users will not be creating permanent accounts. Thus, a database storing user information is not a consideration.
* The library of images is expected to increase for Draw It or Lose It.
* The current hardware requirements for the application's operation are hard to estimate, so rather than dive into specific hardware recommendations, *the server's hardware requirements should quadruple the current hardware requirements for the application except for storage requirements.*
* The game's development will be over after it's completed, except for minor patches. After that, the server will be dedicated Draw It or Lose It.

The server is expected to be relatively static regarding the additional information to be added to the database. The current library of images uses less than 1.5GB of data. A typical Linux installation is between 4GB and 8GB, but for future scalability, assume that the operating system size will multiply significantly. Thus, 32GB should be allotted for OS space. The library of images could be considered user files, so allocate 32GB of space for user files (including the library of images for the application). Solid-State Drives are the quickest file-accessing method currently, so a 64GB SSD will be more than sufficient for the server's purposes. The significant investment will come in the form of RAM available. The server will also be acting a middle-tier, which means all the processing logic will happen before being sent to the client. Thus, the middle-tier will be holding a large amount of information in memory, such as the current users of a game, the image that the game is rendering, the potential answers for a team, and pretty much everything except for the users' interactions with the solution they choose and UI elements. However, memory requirements will be discussed in the following section.

1. **Memory Management**: **Cloud Computing** or **64GB DDR5 RAM**

Linux is a vague name for numerous UNIX-based systems. It depends on the variant of Linux installed on the server as to how much memory the operating platform supports. Most 32-bit operating systems support a max of 4GB or 64GB of RAM, depending on what features are enabled. 64-bit operating systems support significantly more RAM, up to 256TB of RAM. With the variability between the systems and a lack of information about how many users Draw It or Lose It currently has, the best method of RAM utilization would be to use cloud computing. Since cloud computing offers flexible amounts of RAM, there's no need to consider how much RAM the server will need because cloud computing will provide for RAM as required. Essential factors to consider on a programming level for memory management are opening and closing files. It is vital to free up RAM by releasing images that the program is no longer used.

1. **Distributed Systems and Networks**: **REST API**

REST stands for representational state transfer. REST API is a sensible choice in this model for distributed systems because it expands, or rather enables, the functionality intended by choosing a multi-tier architecture. REST is how the server can pass data to the middle-tier, the middle-tier can do the processing logic (such as the rendering of the image), and the snippet being rendered can be passed to the various client machines, which in this case, are browsers. However, there are a few concerns to be wary of, specifically in the exchange between the middle-tier and the client.

1. Security – Design the system so that malicious code cannot intercept the transmission from the middle-tier to the client
2. Stability – While the server OS is typically stable, if there is an issue happening with one of the browser platforms, that can stop a substantial amount of users from getting to play, which could result in a loss of returning users
3. Connectivity – Using a REST API and transferring everything between the middle-tier and the client requires a strong internet connection. So, one of the primary concerns for playing the game is ensuring each user has a solid and stable internet connection.

Linux does enable REST API to execute, and various software options permit the process. In addition, Linux is secure and stable, so it addresses a couple of the concerns from the server and middle-tier side of the application. Thus, utilizing Linux can mostly eliminate concerns about outages due to server failure. However, client security and stability are beyond the application's capabilities.

1. **Security**: **Design safe software**

Since this software design document assumes that user information will not be saved and they do not need to enter sensitive information to play, securing user information on the server-side is not a significant concern. The primary concern for security lies in the program's design and the data transfer between the middle-tier and the client. Therefore, the server and middle-tier should never be receiving code, only information in a text format. Information should also be cleaned and sanitized (this means ensuring that code is not being transported via text) before entering the code that the middle-tier will execute after receiving a response. The game should not request more information than it needs to operate from the user, which minimizes the risk of sensitive user data being captured by malicious programs or hackers. The application should also inform users what information from the user and the device it *collects* to ensure users know what information is at risk. Another security concern is that the server and middle-tier are not transferring malicious code. Linux is traditionally secure and not typically the target of cyber attackers and viruses because most individuals using Linux are more technologically proficient, and Linux is not a widely used OS. Still, the server and middle-tier should routinely be rebooted to ensure that the system is not compromised. Thus, the server and middle-tier should have firewall and antivirus software installed to detect potential breaches.