

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## 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 | 05/16/24 | Scotty Folker | Provided Requirements Given By Client |
| 1.1 | 05/26/24 | Scotty Folker | Updated Revision |
| 1.2 | 06/09/24 | Scotty Folker | Updated Evaluation |
| 1.3 | 06/23/24 | Scotty Folker | Finalized Document |

## [Executive Summary](#_sbfa50wo7nsh)

Our company, CTS has been tasked with designing an Android based game called Draw It or Lose it, inspired by the 1980’s game show. This game’s design is based on their website version of the game and must be structured accordingly. The game consists of teams competing against one another to guess the subject of rendered images from a stock drawing library, with a number of players per team. Using the Singleton design pattern, the game instances must happen one at a time. It will adopt a client-server structure with a required internet connection to maintain the game among the users. These elements are critical to the game’s functionality as well.

## Requirements

**Business Requirements**:

* Create a usable UI that allows teams to compete and guess the images using given clues.
* Allow for unique naming of teams and games to prevent conflicting parameters.
* Create a system that incorporates scoring and tracking for each team and player.
* Allow for a single game instance in which multiple teams and players participate.

**Technical Requirements**:

* Implement appropriate security measures to ensure secure communication and data transfer transfers between clients and the server.
* Design a client-server structure with a centralized game server to handle the game logic and manage client interactions.
* Employ a distributed system structure to manage multiple concurrent game instances and client connections.
* Ensure cross-compatibility by developing a web-based application that can be accessed from a variety of devices and platforms.
* Promote code efficiency through the use of object-oriented programming principles to allow for the ease of maintainability.

## [Design Constraints](#_2et92p0)

In order to develop this game and have it run on many different devices, some important challenges must be addressed:

1. **Working on Multiple Devices:**

* Game must be able to function on different devices, like Android, iPhone and PC.
* Test extensively to ensure functionality across these platforms.

1. **Required Internet Access**

* Game must have a reliable internet connection to ensure that players can participate without issue.
* Implement exception handling to prevent network failures.

1. **Interactivity**

* Must have real-time interactivity between players and the server.

1. **Private Access to Data**

* Protect sensitive player data via encryption.

1. **Allow For Game Growth**

* Allow for a scalable structure to ensure the game does not get overloaded and fail.

Addressing and conquering these challenges will allow for functionality across every platform, provided that different development frameworks are involved and can address these challenges directly.

## [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 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**  **Software Reliability:** Web-based games can be more reliable, provided the proper libraries and software.  **Cons**  **Expensive hardware:** Parts for the platform tend to be expensive to replace and maintain  **Restrictive User Interface:** Platform doesn’t allow for much freedom when it comes to the usability of the software. | **Pros**  **Secure servers:** Due to Linux’s open-sourced nature, servers that host software tend to be more secure.  **Independent servers:** Linux tends to not rely on outside sources in order to function.  **Cons**  **Tech Expertise:** Linux tends to appeal more to those who have more expertise within the given distribution when it comes to setup and maintenance. | **Pros**  **User-Friendly UI:** Windows has always been a preference for a lot of users, given its ease of use among users.  **Widely Used Software:** This can be a big plus when designing software, due to the mass use of the same software among users around the globe.  **Cons**  **Security Risks:** Windows has had its fair share of security breaches, mainly due to its susceptibility to hacking and its reliance on outside servers and resources. | **Pros**  **Browser gaming access:** Mobile devices allow for their web-based games to function within the browsers of cell phones and tablets.  **Specific Applications:** Mobile devices can serve users well by all their users having access to the same application being used to make the game function.  **Cons**  **Limited Battery Life:** As opposed to having an outside power source, such as with a laptop or desktop, a mobile device can only be used for as long as its battery will allow before needing to be charged. |
| **Client Side** | **Pros**  **Testing Software:** Mac has access to many different forms of software that allow for outside testing to ensure proper functionality among other platforms.  **Relatively Low Development Time:** Due to Mac’s ease of use in terms of development, it can take less time and even be less frustrating than Windows due to its familiarity among other Apple software.  **Cons**  **Extremely Limited User Access:** Mac requires you to have an Apple product in order to use any of its MacOS software. | **Pros**  **Wide Range of Libraries:** Linux allows for hundreds of thousands of libraries that are more than capable of handling different client types.  **Low Development Time:** Linux can provide very user-friendly development access that can greatly limit the time spent developing software.  **Cons**  **Expertise Needed:** As mentioned, Linux requires (to an extent) a certain level of expertise in the software to use properly and effectively. | **Pros**  **Widely Used Software:** Windows happens to have the most popular development tools on the market due to their ease of use.  **Not Much Expertise Needed:** Unlike Mac and Linux, not much expertise in the software is needed to develop effectively.  **Cons**  **MacOS Browser Functionality Limit:** It’s a bit more difficult to test software for the MacOS platform browser. | **Pros**  **Easy-to-Use Tools:** Mobile development apps tend to be pretty effective to utilize when it comes to web-based games.  **Cons**  **Expertise Needed:** A certain level of expertise is needed to make these tools useful.  **Longer Development Time:** Mobile tools can add more time to development, due to the specific needs of mobile devices, such as touch-based interactions. |
| **Development Tools** | Common languages include:  **Java**  **Python**  **Ruby**  **Javascript**  **Pros**  **Apple Review:** Software can be reviewed by Apple to ensure good content and security.  **Virtual Machines:** It’s relatively easy to run Windows or Linux on virtual machines. | Languages include:  **Java**  **Python**  **Ruby**  **Node.js**  **Open-Sourced IDE’s**  **Pros**  **Anytime Deployment:** Linux does not require authorization from an outside source, due to its open-sourced nature.  **Virtual Machines:**  It’s relatively easy to run Windows or MacOS on virtual machines. | Typical languages include:  **Java**  **Python**  **C#**  **.NET**  **VS Code**  **Pros**  **No License Cost:** It does not require license purchase or authorization to use development software.  **Virtual Machines:**  It’s relatively easy to run Linux or MacOS on virtual machines. | Languages involving mobile devices can include:  **Java (for Android)**  **Xcode (for iOS)**  **Swift**  **React** |

## 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 I prefer Windows, the operating system that would be best for this application would be Linux for a variety of reasons. For starters, it would definitely be cheaper, given the limited licensing costs and doesn’t have limited access to data centers like Windows. Linux also has good security when it comes to personal data, as it is open-sourced, unlike macOS and Windows.
2. **Operating Systems Architectures**: Since low latency is not required for this game to function, as it does not rely on fast-acting responses, a backend server that manages the environment with frontend rendering would be the best choice of architecture.

To allow for scalability to tackle the challenge of game growth, we can use Kubernetes or Docker to run a more modern backend running containerized microservices.

If we use frontend for rendering, we can offload some of the more resource-intensive aspects of the game to reduce data center costs further, as mentioned before. We can also implement some kind of pre-rendering, that is, allow for caching of subsequent images beforehand to ensure framerate sustainability.

1. **Storage Management**: The recommended storage type ought to be solid state drive to ensure that the game runs smoothly and allows for data allocation, although hard drives ought to work just fine as well.
2. **Memory Management**: I appreciate Linux’s concept of page cache for data stored in both main memory and virtual memory for any pages allocated. Linux allows for its memory to not be constantly filled with data it’s not currently using, as it incorporates the LRU algorithm, or Least Recently Used algorithm.
3. **Distributed Systems and Networks**: We can accomplish the goal of allowing players to play on different platforms at one time by having different aspects of the game running on different systems connected through the same network.

One server can maintain the game state that also manages all of the rules and players. Each player has an app running on their device that acts as a client app to communicate with the server. One player makes an action, that action is communicated to the server and the data is received and processed. It then updates the game for the rest of the players. This allows everyone to play together without issues.

This brings up the issue of the internet connectivity requirement. If it fails, the server will not update the game for the players. Therefore, there needs to be backup instances to address this possibility.

1. **Security**: There must be many different security measures to ensure that players remain protected from outside forces that wish to obtain sensitive data, including:
   * + 1. Firewalls to control network traffic.
       2. Encryption for sensitive data.
       3. Authentication for users like usernames and passwords.
       4. Automatic updates for clients.

As well as other methods of security. These features are prevalent among both Linux and Windows, even though Linux comes with less risk of security breaches due to its inherent open-sourced nature.