

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 | 03/26/2025 | Rebecca Eaton | Java script, executive summary completed |
| 2.0 | 04/06/2025 | Rebecca Eaton | Server and clients side breakdown, developmental tools |
| 3.0 | 04/19/2025 | Rebecca Eaton | Explain OP architectures. Analyze the differential file system components that represent a collection of data. Explain the functions of memory and storage management. Identify memory management techniques. Describe the distributed system |

**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)

We aim to expand your “draw it or lose it” Android game. We are trying to create a web-based application that can be used across multiple platforms to help widen the game's audience and improve accessibility.

## Requirements

We will work on creating a platform that is a web-based version of the game to ensure compatibility across different devices and platforms, to help user engagement through a better interface, and improve the game's memory to manage the library stock of drawings on hand.

## [Design Constraints](#_2et92p0)

The main issue is to ensure we have real-time interaction between players across different platforms, and it would require efficient servers to be able to handle multiple games running at the same time, and to ensure there is no lag with the data needing to be transmitted through different platforms, and just the cross-platform compatibility.

## [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)

There are multiple classes such as Entity, Game, Team, and Player.

**Object-Oriented Principles:**

* **Inheritance** is used to avoid code duplication, allowing **Game**, **Team**, and **Player** to share common properties through **Entity**.
* **Encapsulation** ensures that game state is managed internally, exposing only necessary methods for interaction.
* **Aggregation** represents the "has-a" relationship between games and teams, and teams and players, organizing the model into a coherent structure.

Class relationships:

* Game inherits from the Entity class and then aggregates from the Team class to help the game have multiple teams at once
* Team inherits from entity as well and aggregates player class to have each team consisting of multiple players

**"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** | Characteristics:  Proprietary software, Sleek, user-friendly interface, Integrated with other Apple devices  Advantages:  High level of security, Excellent customer support, High-quality proprietary software  Weaknesses:  Expensive, Limited customization options | Characteristics:  Open-source, highly customizable, Command-line interface  Advantages:  Free to use, High level of security, Wide range of distributions  Weaknesses:  Requires technical knowledge to be used effectively, Limited support for games and proprietary software | Characteristics: Widely used, Compatible with most software, User-friendly interface  Advantages:  Wide range of software compatibility, Regular updates, Large user community for support  Weaknesses:  Vulnerable to viruses and malware, can be slow and prone to crashes | Characteristics:  Portable, Touchscreen interface, App-based  Advantages:  Convenient and portable, Wide range of apps, Integrated with other devices (e.g., smart home devices)  Weaknesses:  Smaller screen size, Limited processing power compared to desktops/laptops, Battery life can be a limitation |
| **Client Side** | Pros:  •Easy cross-  browser testing  software  •Medium  development  time and  deployment  Cons:  •Requires Apple  product with  MacOS | Pros:  •Tree files  structure  accommodation  •Works with  every web  browser due to  plethora of  open-source  software  •Quick  development  and  deployment | Pros:  •Easy cross  platform  testing besides  MacOS  •Quick  development  and  deployment  Cons:  •More difficult  to test for  MacOS  browser. | Pros:  •Expertise with  android app  development  Cons:  •Difficulty  testing other  environments  and browsers  •Longer  development  time |
| **Development Tools** | documentation  for deployment  on MacOS  •Can easily run  Windows and  Linux through  local Virtual  Machines  •Application  must be  reviewed and  approved by  Apple  •99$/year for  Apple Dev  program  •Mac OS X  •Eclipse for Jav | •Deployment at  any time  •Can easily run  Windows and  MacOS  through local  Virtual  Machines  •Shell prompt  and terminal  •Open source  community  •No license cost  •Eclipse for Java | •Deployment at  any time  •Extensive  documentation  for deployment  on Windows  •Can run Linux  through VM  •No license cost  •Visual Studio  Code for  Javascript/HTM  L  •Eclipse for Jav | Xcode 12 for  deployment  to iOS  •99$/year  Apple dev  program for  iOS  •SwiftUI |

## 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**: Out of all the platforms we went over, I would select the Windows operating system due to its ability to use the current Android build and convert it into a cross-platform game. Windows allows you to design, develop, and open apps easily and inexpensively, and it may be the largest user-based operating system right now, meaning most people already use Windows, so it would be available to most people.
2. **Operating Systems Architectures**: Windows separates its operating system into User mode and Kernel mode. User mode processes user-facing and affects much of what the user interacts with. Kernel mode is more under the hood and low level, dealing with inputs and outputs, memory management, networking, hardware management, and routines.
3. **Storage Management**: for storage, I would suggest Microsoft Azure due to their competitive prices, constant updates and support, and customer support. They have cloud-based storage and have storage options including Azure File system, Azure Storage Containers, and Azure Blob Storage
4. **Memory Management**: Windows had a disc paging and demand paging storage system that acts as an extension of the computer's physical memory.
5. **Distributed Systems and Networks**: Azure offers max uptime with cloud-based insights, email alerts, and monitoring solutions, and this will help when we scale up the game to 4 players in each game.
6. **Security**: IP configurations for whitelist specific access to resources (like player or personal information) or the entire app. The option for storage in a VPN within the cloud for enhanced security. Database could be IP Whitelist access only, have a password, and require SSL connectivity to protect the user data.

**Server-side advantages and weaknesses:**

* **Mac**
  + **Advantages:** compatible with IOS and has good development environments for ease of use.
  + **Weaknesses:** higher cost for production of hardware and limited scalability.
  + **Licensing costs:** Mac is meant for and requires Apple hardware, which, if not already in use, can be costly, but there are no additional fees for the operating system itself.
  + **Server-based deployment:** Mac can host web applications, but it is not meant for or optimized for it. Mac is not typically used for any large-scale server development.
* **Linux**
  + **Advantages:** strong community support and a wide range of software and tools in its OS, as well as being known for its high performance.
  + **Weakness:** requires someone who knows Linux well to manage and configure the OS in our game.
  + **Licensing costs:** thankfully, Linux is an open source, so there are no direct costs for licensing, but if any support or maintenance is needed, there may be additional costs.
  + **Server-based development:** Linux is known for its stability, flexibility, and security. It is also known for being able to support various web services like Nginx, Apache, and many more.
* **Windows**
  + **Advantages:** integrated with Microsoft servers and products and has a user-friendly interface.
  + **Weakness:** In comparison to Linux, it has a higher cost and potential security vulnerabilities.
  + **Licensing costs:** Windows requires licensing, which can become expensive due to the version of Windows needed and how many of users we have.
  + **Server-based development:** Windows Server is a popular choice for hosting web applications, especially in enterprise environments.

**Client-Side Evaluation:**

* **Desktop:**
  + **Compatibility:** We should use web technologies such as HTML5, CSS3, and JavaScript to help ensure compatibility across all desktop browsers.
  + **Development Considerations:** A responsive design is needed to make sure the application works on different types of devices due to their screen size and resolutions.
* **Mobile:**
  + **Compatibility:** We need to work on optimizing the application for mobile browsers and ensure compatibility with both iOS and Android devices.
  + **Development Considerations:** Testing on multiple devices and screen sizes is necessary. Consider using frameworks like React Native or Flutter for cross-platform development.

**Development Tools**

* **Programming Languages:** JavaScript (with frameworks like React, Angular, or Vue.js), HTML5, CSS3 for front-end; Node.js, Python, Java, or PHP for back-end.
* **IDEs and Tools:** Visual Studio Code, WebStorm, Eclipse, or IntelliJ IDEA for development; Git for version control.
* **Impact on Development Team:** A special skill set is required to handle both front-end and back-end development. Multiple teams may be needed to manage different aspects of the application.
* **Licensing Costs:** Many development tools are open-source or have free versions, but some enterprise tools may require licensing.

1. **Operating Platform:**I suggest using Linux because it is very versatile. It is a customizable operating system that strongly supports a wide range of programming languages and tools to help make development easy and more user-friendly.
2. **Operating Systems Architectures:** Three operating architectures are primarily used, such as **Hybrid Architecture,** which is a combination of the other two platforms that I will describe shortly, where system services are built as monolithic kernels and have separate modules. **Monolithic Architectures,** are used by all system services where the user services run in the same space but this can cause system crashes although it will give us high performance which is why we would need to use some **Microkernel Architectures** to help use avoid the issue of the system crashes microkernels divides the system into smaller components that run in different address spaces to help reduce the issue of system crashes again. Linux mainly uses a monolithic kernel design, but they do partially use microkernels since it has a flexible architecture to help us in the multi-user and multitasking capabilities that are needed.
3. **Storage Management:**I would recommend using a distributed file system since we need scalability and the potential for many users generating content at the same time. This would help to allow the game to distribute and replicate data from and across multiple servers like Ceph and GlusterFS, which would help us ensure data is available and there are no faults in the gameplay.
4. **Memory Management:**Linux offers memory management that is very efficient for games. It has virtual memory and demand paging that optimizes where the physical memory goes and will help ensure memory fragmentation is minimized. We would also need to add application-specific optimizations to ensure proper performance.
5. **Distributed Systems and Networks:** We will need to implement a distributed system that consists of a central game server and allows multiple client devices. We will need to ensure they can communicate properly over a reliable network, so using something like a RESTful API would help us get better communication between the client and server. For network connectivity, I would recommend using a load-balanced architecture that helps distribute traffic across multiple game servers, like CDNs, to help us provide scalability and fault tolerance within the game.
6. **Security:**When it comes to security, there are a few things I would recommend, like data encryption to safeguard user information. We should also investigate adding strong access controls and authentication options, like passwords or multifactor authentication options for an extra layer of security. We should also ensure we regularly update the server to check and address security breaches and fix them as soon as possible. We would perform security audits often and penetration testing to help us identify any potential vulnerabilities.