

<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 | <1/26/2025> | Audrey Saidel | Completed executive summary, requirements, design constraints, and domain model |
| 1.0 | 2/9/2025 | Audrey Saidel | Completed evaluation |
| 1.0 | 2/22/2025 | Audrey Saidel | Completed 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)

Creative Technology Solutions (CTS) has partnered with the Gaming Room to help create a web-based version of their android-only application “Draw It or Lose It”. The Gaming Room has asked for CTS assistance in setting up the environment as well as streamlining development across multiple platforms. The Gaming Room has provided us with a list of software requirements to follow to stay true to their original application while also seeking to expand their user base.

## Requirements

**Business Requirements:**

* Streamline the development process for maximum progress efficiency

**Technical Requirements:**

* The game should be accessible to multiple platforms (operating systems + devices)
* The game should retain functionality from the original android application while following software requirements
* Assist staff with setting up the necessary development environment for supporting a web-based application

## [Design Constraints](#_2et92p0)

* Creating a multi-platform web-based application will require a large amount of testing time as it must span many different devices and operating systems (Req 2)
  + Testing needs to be done on different browsers
  + This might make it difficult to streamline and create an efficient work plan, will create a longer development time in general due to the number of tests (Req 1)
* Retaining functionality and design can become a constraint as UI changes from an android interface to a web based one. (Req 3)
  + Inputs such as touch have to change to support a mouse.
  + Not only this, but UI must scale to fit different devices, such as a Safari browser on an iOS versus Microsoft Edge on a PC.
* Acquiring the original code and adapting it for the new web-based environment could also prove difficult (Req 4)

## [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.**

The UML diagram above displays seven classes: Entity, Player, Game, Team, GameService, and outside of the core object model, ProgramDriver + SingletonTester.

Game, Team, and Player all inherit from Entity, resulting in Entity becoming a superclass with three separate subclasses. Team and Player are associated with a zero to many multiplicity, as one team could have zero or many players. Game and Team are associated with a zero to many multiplicity as one game could have zero or many teams. Game is associated to GameService with a zero to many, as GameService could have zero or many games. Outside of the core object model, ProgramDriver uses SingletonTester to ensure that only one instance of GameService can be created.

Encapsulation is displayed in the core object model through the use of private variables and public accessor methods. For example, the GameService class stores a list of games privately, and it can only be accessed via accessor and mutator functions, like addGame() or getGame(). This way, data is protected, and access is controlled. The accessors and mutators in each class also utilize abstraction by creating ways to access the private variables without worrying about the data underneath. For example, in the Team class, there is a private list of Players. Users can mutate this private list via the mutator method “addPlayer()”. Polymorphism is used during both runtime and compile time. During runtime, toString() in the Entity class can be overridden by toString() in its subclasses Player, Team, and Game. During compile time, function overloading occurs in the GameService class when getGame is overloaded with different parameters.

## [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.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Apple discontinued MacOs server in 2022, so MacOS is no longer optimized for large scale server development. Built on UNIX similar operating system, increasing stability Local hosting is available using built in UNIX-based tools. Many server features are still available but difficult to configure. Not widely used for web hosting overall. | Linux is free, open source, and the most commonly used for web servers. It is also the preferred choice for cloud servers, such as AWS, DigitalOcean, and GCP. Multiple different distributions, like Ubuntu (heavily supported by Cloud Servers) or Debian (best for its feature set stability as well as its security). Linux is highly regarded for its customization and optimization benefits, leading it to become industry standard for servers. | Windows server license is among the most expensive of any server operating system. The exact price fluctuates as there are different editions and license models. The common range listed on their website is around the 1,500-6000 dollars per year area. The best advantage is its integration with Microsoft technologies like ASP.NET or MS SQL. It is also known for its scalability and flexibility. | The backend will have to be hosted on one of the other operating systems.  It is not practical to host a server on a mobile operating system. However, deployment costs for the Apple App Store is $99 dollars per year. |
| **Client Side** | Supports major web browsers and includes Safari. If a Mac is already owned then MacOS doesn’t cost anything, but if not, you will have to pay. Apple machines are very expensive, in the thousands range. UI design benefits, the UI is minimalistic and widely praised. Developers should have expertise in iOS development, using built-in Apple tools, and knowing what other development tools are compatible. | Compatibility with all major browsers. Multi-platform client-side development tools are commonly free and open source, so costs will remain low in that aspect. Most people do not use Linux as their day-to-day computer operating system. Developers should already have expertise in Linux coding as there is a steep learning curve. If the developers are not already familiar with Linux, it will take more time as it is complex. | Supports major browsers. Web-development will be fast paced. Windows PCs come at a lower price point than mac. Not a UNIX type system so it works differently than macOS or Linux. Not as frequently used for web development. No matter what, software will have to be installed. | Runs mobile browsers like iOS Safari and Android’s Chrome. It has native iOS and Android apps respectively. Slow development time if not doing cross-platform and instead making separate Android and iOS environments. These development tools are free, but when the app is set to be launched there are fees (see box above). Cross-platform development is faster but may involve outside costs for different frameworks. |
| **Development Tools** | It has free built-in development tools from Xcode, which is a heavier IDE. VSCode is more lightweight and can also be used in MacOS. VSCode is extremely customizable with a large extension library. Supports languages like Python, | Excels in server-side development, as well as cloud and backend development. Supports Git, VSCode, Nvim, Docker and more. Jetbrains products are good for IDE’s but are costly. Due to the nature of Linux programming, it is considered very lightweight. | Many Windows exclusive features. Visual Studio IDE is at base level free but does come with added expenses if better versions are desired. VSCode and Eclipse are also commonly used IDEs. Supports .NET, C++, Java, Python, and more. Best in Game and Enterprise development, still behind Linux when it comes to server development. Offers PowerShell for an experience similar to Linux. Git bash terminal can be downloaded. | Android Studio can be used to create separate environments for iOS (Swift) and Android (Java/Kotlin) devices. It comes with a built-in android emulator. Xcode can also be used for iOS development. Can use React Native and its JavaScript libraries for cross-platform development as well. Flutter is like React Native but performs better as it compiles to native code and React Native relies on native modules. However, Flutter uses Dart which is uncommon. Both need emulators. |

## 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**: The best operating platform for The Gaming Room to utilize is Linux with some cloud-based implementation. It would be best to use the Linux Ubuntu server as described above, it is heavily supported by cloud-based server hosting. The recommended cloud-based platform to use with Linux would be Amazon Web Services (AWS), as it is the most widely used, cost efficient, and flexible system for this project. Using both Linux (an open-source OS), and AWS will provide a cost efficient, but still high-performance multi-platform deployment. All storage will be cloud based, but backend will all be hosted on the Ubuntu server run on Amazon EC2.
2. **Operating Systems Architectures**: There are two main architectures to consider, the client side and the server-side architecture. Each of these will need to have their own methods of deployment.

**Front End Architecture:**

The application will run on Android, iOS, and web applications. Each platform will have separate clients that interact with the backend through REST API.

*\*\* It is also important to note that it would be helpful to move mobile development to React Native to ensure a consistent codebase between the current Android app and our future iOS app.*

**Backend Server Architecture:**

1. *Amazon EC2***:** This is to create an instance of the Ubuntu server on a rentable computer. This will be where our server runs, from a secure data center. Intended for long term operations (why we are not using AWS lambda), allows for full control over the software and OS (acts as a virtual machine). This full control can allow us to download cross-platform necessities and custom configurations. Network and security will be installed by us.
2. *Ubuntu Server:* Ubuntu is a Linux distribution that is known for its compatibility with cloud hosting, security, and stability. Compatible for communication with our RDS database. Will have to be updated manually.
3. **Storage Management**: There are many useful types of storage management that should be utilized for cloud storage. As discussed in the storage consideration section of my memory and storage analysis, utilizing cloud storage would be optimal for the scope of our project. This is due to a reduction in 24/7 maintenance, security issues, and unnecessary costs. As we are using AWS Cloud for our operating platform, it is best to utilize their different forms of data storage and the ones that are most optimal for our variety of data that needs to be stored long term.

**Recommended Cloud Storage Utilization:**

1. *Amazon S3:*This is the standard and simple AWS plan. It offers scalability, reliability, lowest price, and high performance. We would utilize this type of cloud storage to store the 200 HD images for the game. There is 99.99% availability guaranteed, and the scalability allows for more images to be added over time if necessary.
2. *Amazon Relational Database Service (RDS):* This is the optimal way to do cloud-based database storage. This database would be used to hold user data instead of using a hash map in the main memory. This database service offers availability, choice of engine, and reliability.
3. *AWS S3 Glacier:* Although not mentioned in the memory and storage paper, Glacier is an important asset for storing long-term data. The glacier can be used to store users that have been inactive for a certain number of months without deleting their data or clogging the database. It could also store old game logs for up to a certain number of months in case moderators need to search for game data (in the case of cheating or a number of other in game offenses).
4. **Memory Management**: Memory management is essential for avoiding crashes, lag, and other overflow issues. First, it is important that we stop storing user data in main memory and instead swap it to secondary memory. Linux servers come with many different memory optimizing techniques.

**Recommended Memory Management:**

1. *Swapping:* Swapping is a memory management technique that moves processes from the main memory to secondary memory. Inactive data (such as inactive games) can be stored in secondary memory (Glacier) rather than clog up memory. Swapping allows for user data to be retrieved when a user is actively playing as well. Excessive swapping can hurt performance, so use it sparingly.
2. *Paging:* Paging (or Pagination) is a memory management technique that can divide memory into blocks (pages) and retrieves them from the secondary storage into main memory. This can be used to minimize image load time. This is already enabled by default in Linux.
3. *Linux Specific Memory Compression:* Zram can be utilized by Linux and EC2, like swapping but instead of going to storage, uses CPU for compression. It has faster swap time and would be better for the mobile devices as they have limited RAM.
4. *Memory Cleanup:* The most important memory technique for our program is memory cleanup. Because we are using java, we can utilize garbage collection which works in the background to delete unused objects from memory.
5. **Distributed Systems and Networks**: A distributed system consists of an application distributed between multiple devices, and the network that connects the devices. Our multi-platform game will continue to utilize REST API for client-server communication. This is because REST API allows the different clients to connect to the same server through HTTP requests. When it comes to dependencies and dealing with connectivity and outage issues, AWS elastic load balancing can help ensure high availability by distributing traffic across multiple instances. However, it is unlikely that EC2 will fail, and if it does it is likely due to overusing resources such as RAM or CPU.
6. **Security**: Having control over the server means that we need to implement our own security on it. There are some options we can utilize through AWS, and others that are more traditional and need to be done ourselves.

**Server Security**

1. *AWS Shield:* Helps prevent DDoS attacks, especially important in competitive games. Standard version is already included.
2. *AWS WAF*: Web application firewall that monitors end user HTTP requests. Filters and monitors this traffic to prevent attackers from exploiting.
3. *RBAC:* Role based access control that is already implemented. Different roles for players, admin, etc. Different roles have only the permissions they need and no more (enforce least privilege). Admins should have access to backend data, while players can only access gameplay functionality.
4. *Input Validation:* To prevent common web-application injection attacks (XSS, SQL), implement input checks for usernames, passwords, and any other input.
5. *Regularly Update:* Linux OS and our application should be patched and up to date. AWS provides updates but does not install them for you on EC2.
6. *Data transmission:* Data transmitted between the client and the server should be secure. HTTPS through REST API can help prevent eavesdropping and tampering of data. Uses SSL and TSS encryption.

**Storage Security**

1. *Encryption*: AES-256 encryption to encrypt sensitive data inside RDS database. Even if the database is not compromised, it’s important that the data always stays protected.
2. *RBAC:* Role based access control again to control who has access to what parts of the database. Multi factor authentication should be used for admins with the most access.