

<Draw-It-or-Lose-It>

# **CS 230 Project Software Design Template**

Version 1.3

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc141615942)

[Table of Contents 2](#_Toc141615943)

[Document Revision History 2](#_Toc141615944)

[Requirements 3](#_Toc141615945)

[Design Constraints 3](#_Toc141615946)

[System Architecture View 3](#_Toc141615947)

[Domain Model 4](#_Toc141615948)-5

[Evaluation](#Evaluation) [5-8](#Evaluation)

[Recommendations](#Recommendations) [9](#Recommendations)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <7/14/2023> | <Logan-Taylor> | <Initial development of Software Design document.> |
| 1.1 | <7/16/2023> | <Logan-Taylor> | <Expansion of development requirements section.> |
| 1.2 | <7/29/2023> | <Logan-Taylor> | <Modified UML diagram to accommodate changes.> |
| 1.3 | <8/13/2023> | <Logan-Taylor> | <Expansion of recommendations for the client.> |

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

CTS (Creative Technology Solutions) has requested we develop a web-based version of their gaming application, Draw It or Lose It. To develop this app, we must implement a round based system reliant on 2-minute rounds, image rendering over a set period (30 seconds) from a library of stock images, a player-based team system and point tracking for both individual players and collective totals for teams.

## Requirements

Full functionality of requested gaming features, image rendering/revealing smoothly over time, randomization (seed) based answers, and real-time round timers.

## [Design Constraints](#_2et92p0)

Team composition: Depending on the route the client pursues, expertise on specific languages, operating systems, environments, and associated tools (such as .NET Frameworks.)  
  
Finances: With a variety of options and price ranges, more information on budget is required to determine which routes are suitable for the client.  
  
Scheduling: In tandem with financial considerations, a timeframe should be addressed as early as possible. Such constraints may influence advising, viability of the project itself, and allocation of resources.  
  
Licensing: If the client pursues Windows based web development, among other less widely known software restrictions, the sooner that we know this information the sooner that we can navigate such hurdles.  
  
Hosting: More information is required from the client to better advise, however, such a concern should be addressed early into development.  
  
Security: While such a project does not inherently hinge on security, protection of company data and users should be addressed. Options such as Windows may lack in this department, while exceling in other areas, where Linux offers additional protections that should be brought to The Gaming Room’s attention.

## [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)

To set up this application a series of classes should be set up, below is a UML diagram of the suggested classes and the relationships between them.

**A diagram of a computer

Description automatically generated**

First, we have ProgramDriver, a class designed to ‘run’ the application, the interface that users will interact with is controlled from here. ProgramDriver uses SingletonTest to test whether an instance is ‘Singleton’ in nature, which simply means that there is only one instance active at any given point. This prevents data from being created multiple times without compromising the encapsulation of the Game/GameService classes. This relationship is simply for testing purposes and bug fixing should the need arise.  
  
Next, we have the Entity class, a class that can be viewed as a ‘save’ or currently active game class. Note that GameService, Game, Team, and Player all use inheritance from Entity, this helps organize and reduce redundancy when working with many object and object types.  
  
GameService, while ProgramDriver runs the application, runs the game itself. It oversees adding games, retrieving games, managing players, managing teams, and querying all the aforementioned objects while maintaining an id system to prevent overwriting data. Game is connected to GameService, storing information for save objects that GameService uses. While Game needs GameService, GameService has a 0 to many relationship with the ability to instantiate new Game objects.  
  
 The Team class is similar in nature to the Game class, while instead of ‘save’ data being stored, Team stores player data. Team also contains a public method for adding to an object’s ranks. Team has the same relationship to Player as GameService does with Game, however, the data is sent back to GameService for manipulation instead. Team also has a relationship of many to 0 with Game, allowing many teams to exist for any given Game, should it exist.  
  
The Player class is simple, it simply stores information for a player with a class object. These players are stored in Team objects, which can be queried by GameService.  
  
The game data, including teams and player data, uses encapsulation to prevent game data from being copied in memory multiple times, prevents game data being altered when it is not appropriate, and compartmentalizing data structures where appropriate. A notable example is tying data for players to teams, while encapsulating the ‘parent’ objects to allow querying/manipulation via specific games or teams.

**Evaluation**

See pages 6-8

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac computers provide excellent security with user-friendly and smooth design. Mac is also widely used by developers, with many software developers having creative experience using it.  Macs have a Unix-based operating system, which make them compatible with many web development tools and frameworks. MacOS provides a stable and simple environment for web hosting.  However, Mac based hosting tends to have a higher price point and are limited regarding hardware compatibility compared to options such as Windows. | Linux is an open-source operating system, with a wide variety of distributions such as Ubuntu. Linux also offers a high level of customization and flexibility, making tailoring systems to fit client needs easier.  Linux operating systems is highly scalable, providing a suitable environment for a multitude of resource demands. Linux is widely regarded as stable and secure, provided reliability is a concern, this makes Linux an excellent option.  However, Linux has limited software compatibility/support. Hardware compatibility may also cause issues with certain distributions. | Windows is an extremely widely used operating system, leaving no shortage of developers experienced with it. Windows offers a wide variety of software and hardware compatibility, eliminating concern in that regard.  If our application requires integration, Microsoft offers a wide range of frameworks for development. Many web hosting providers off Windows-based hosting plans, providing a familiar environment for developers.  However, Windows tends to be more susceptible to malware and security threats compared to alternatives. Licensing fees may also harm budget constraints. | iOS and Android operating systems:  Mobile devices are designed for portability and accessibility; mobile devices are also commonly used for accessing web applications such as ours.  Such operating systems offer extensive support for responsive web design, allowing users to use our application on a wider variety of smart devices with their respective screen sizes.  However, iOS and Android operating systems have limited processing power compared to desktop operating systems or server machines. Hosting applications with heavy computational strain may not be suitable for mobile devices, with the possibility of requiring addition resources to develop. |
|  |  |  |  |  |
| **Client Side** | Software development for Mac-based application development tends to have a higher price point compared to alternatives discussed.  Mac may also require additional time to develop to allow developers to ensure compatibility across the multitude of different Mac models, operating system version, and hardware environments.  Generally, developing on Mac requires specific knowledge of macOS technologies and frameworks, requiring expertise that may not be on hand. | Software development for Linux-based application development tends to less costly upfront, with additional advantages such as a lack of software licensing.  Barring extraneous customization requirements, development using Linux operating systems tends to be set at a faster pace. However, some additional time may be spent on ensuring compatibility with software packages available to Linux.  Linux does require expertise in Linux-based development tools, libraries, and command line interfaces. However, such expertise is more common than that of Mac. | Software development for Windows-based application development typically requires licensing for Windows OS and a variety of development tools. Such expenses can be a major factor if budget is a constraint.  Development pace for Windows is among the fastest available due to standardization streamlining development and testing.  While Windows does require expertise in Windows-specific development tools, frameworks, and languages, such expertise is widely available. | Supporting mobile devices can involve additional costs, especially if the client wishes to make their software compatible on both iOS and Android.  As with additional costs, developing with multiple operating platforms such as iOS and Android can add complexity and lead to an increased development time. Developing separate codebases or using cross-platform frameworks may be required, increasing the workload for implementation.  In development with mobile operating systems expertise is a must, particularly developers will need knowledge of languages and platform specific environments unique to both iOS and Android. |
| **Development Tools** | Swift and Objective-C are most commonly used for application development of this nature. Xcode is the official IDE (for both Mac and iOS) development, this can be the only tool needed for such development.  Notably, MacOS is also compatible Java and many Java specific IDEs such as Visual Studio Code. | Linus uses C based languages such as C or C++, and languages such as Python and JavaScript/Node.js, including Java. Visual Studio Code fulfills our developmental needs and offers extensive support for whichever language is decided on.  However, with C based languages, a compiler such as GCC/G++ may be required. | Windows also uses C based languages, and additionally offers support for C# and Java. Windows also opens the opportunity to take advantage of .NET Frameworks, a variety of libraries with pre-built functions and APIs for Windows.  Visual Studio is a commonly used integrated development environments, offering a wide range of support for common languages and .NET Frameworks. | Android development: Java or Kotlin is essential, and familiarity with Android studios environments is required.  iOS development: Swift or Objective-C are languages essential to most iOS projects, and familiarity with Xcode, the official IDE for iOS development, is vital.  React Native is a cross-platform framework that offers development across platforms using JavaScript. |

**Recommendations**

First, given that our client’s request specifies they wish to allow multi-platform use of their gaming app, I would recommend approaching this project using cross-platform frameworks such as React Native to handle mobile development. This would allow us to deploy our code on both Android and iOS without having to code for each platform. Although, this would be for if the client wishes to distribute their application through iOS and Android as actual apps.

However, React Native primarily targets mobile platforms and does not directly support applications for desktop-based platforms. I believe it would be best to use both React Native and Java, to ensure that the compatibility the client desires can be accomplished.

Along with considerations of practicality, some things to keep in mind are:

Responsive design: As we hope to target a multitude of clients on various platforms, it is wise to ensure we cover the variety of resolutions that these platforms offer, particularly with mobile interaction.

Backwards compatibility: With the range of browsers that these platforms provide, it may be wise to develop a graceful degradation for users who use browsers without the latest features and conveniences that, for example, Chrome offers.

Testing: Extensive testing is a must regarding such a project, which may provide an opportunity for advertising. Many game developers allow a limited number of players, usually established online figures, early access to their games. Not only does this provide an additional, and often thorough, avenue of testing, but can help build an audience for the app prior to its release.

Without further detail into the financial considerations our client has, I think it best to develop and host this application on Linux. The inherent added security, flexibility, and reliability provides a general, but solid, foundation for any challenges we may face in the development process.

**Operating platform**:

I believe it is in both our client’s interest and our own to host our server using a Linux platform. Linux provides excellent portability capabilities that would allow The Gaming Room to expand to a broader variety of computing environments.

I believe Linux also offers an additional level of security, especially with the additional layer of security that their kernel (discussed below) provides, that would meet our client’s needs.

**Architecture**:

**Kernel**:

The kernel is a foundational level of design for many operating systems, a quote from InterviewBit’s blogpost “Linux Architecture – Detailed explanation” {1} says it best,

“Kernel is one of the fundamental parts of an operating system. It is responsible for each of the primary duties of the Linux OS. Each major procedure of Linux is coordinated with hardware directly. The kernel oversees creating an appropriate abstraction for concealing trivial hardware or application strategies.”

That said, there are a variety of different types of kernels, however, I would highly encourage selecting an “Exo Kernel” for our purposes. Exo kernels are built to fulfill the needs of our client, as Narang {2} put in their article,

“It is suitable for use when performing application-specific customization. Exo kernels are designed for use in mobile devices. They are a variation of microkernels that include additional features designed for use in mobile devices.”

This specific archetype seems ideal for our needs, although other types of kernels can fulfill our needs, I would consider them to be a less effective option.

**System Libraries**:

Linux’s architecture also includes a library of functions that are implemented within the operating system itself, while being a level removed from the kernel that prevents access right restrictions from intervening with our processes.

**Hardware**:

However apparent, it would be a mistake to exclude the physical foundation that Linux utilizes. As many desktop platforms do, Linux systems are comprised of a variety of peripheral devices, most notably the CPU, GPU, and RAM components. Linux also utilizes a variety of memory storage devices, such as HHD, SSD, and M.2 drives.

**Shell**:

As InterviewBit’s blostpost {1} describes, “Different operating systems are classified as graphical shells and command-line shells.” These ‘shells’ act as a user-end interface between the user and the inner-workings of a system, specifically the kernel and its services. There are two different archetypes of ‘shells,’ a graphical shell and a command-line shell.

**Graphical Shell**:

Graphical shells are what many understand as ‘modern’ computers, providing 2d/3d graphics, user friendly interfaces, and a variety of ‘luxury’ features for users. Applications that always run such as calendars, global (in a local system’s domain) querying, and ‘customization’ of programs such as resizing windows.

**Command-Line Shell**:

Command-line shells are much more utilitarian in nature than their counterparts, often designed around the purpose of starting and switching between tasks. Fundamentally, the major difference between command-line and graphical shells is how they display data. Specifically, command-line displays data as plain text. A good analogy would be using command prompt: while you can technically do just about anything you could normally on your desktop with your command prompt alone, graphical interfacing makes this process accessible to a larger audience for a variety of motivations.

**Conclusion**:

While testing, developing, and general maintenance of our client’s application should take place using graphical shells; command-line shells would provide a more effective hosting platform. While it should be noted that either selection would be sufficient for the task with little setback, our client should be informed of this more obscure option and its implications.

**Storage Management**:

While this is still dependent on the workload and traffic to some extent, as well as the scalability and use of use considerations, a prime candidate for our client’s needs is Content Delivery Networks (CDNs.)

CDNs are designed specifically to deliver content, including images, to end-users with low latency by utilizing caching on servers distributed across the world.

However, local storage is still a good option regarding our client’s current needs. This would simply involve hosting the library of images on our own servers. This does have limitations considering scalability and may increase redundancy.

**Memory Management**:

Virtual Memory:

Virtual Memory, often referred to as “V-RAM,” utilizes virtual memory to abstract physical memory (RAM) from the processes we launch. Each process would be designated to its own virtual address, allowing them to run as if it had access to the physical memory space. This provides excellent memory allocation and helps encapsulate the processes.

Memory Allocation:

Memory allocation is the process in which an application, such as Draw It or Lose It, needs to utilize memory to run, it requests this memory from the operating system. Linux provides memory allocation APIs like ‘malloc’ and ‘new’ in C based languages. The utilization of such APIs ensures the segmentation of memory to help protect it from damage.

Caching:

As touched on earlier, Linux utilizes a variety of caching mechanisms. Some examples would be through page and buffer caching. This would be vital in the smooth operation of Draw It or Lose it considering the importance of images and the possible volume of traffic the app expects to see.

Kernel-Level Memory Management:

Going directly to the kernel development community {3},

“Linux memory management is a complex system with many configurable settings. Most of these settings are available via /proc filesystem and can be quired and adjusted using sysct1.”

While this process has an incredible level of nuance that could be explored, we know that the kernels that Linux uses do require efficient memory management in order to function properly. Kernel memory is separate from user memory and is largely managed using techniques specifically designed for this purpose.

**Distributed Systems and Networks**:

To achieve reliable cross-platform communication for our client’s application in conjunction with distributed systems and networks, we must ensure the design of our architecture can handle dependencies, connectivity, and possible outages.

While this may not be an all-encompassing blueprint to follow, the general direction we will begin with will follow a set of approaches outlined below.

**Client-Server Architecture**:

Our application will be able to use a client-server architecture, where our client’s users (or more specifically the devices of those users) communicate with a central server that will be used as a ‘control room’ for our application which will allow interactions between players and remote access to the game data.

**Cross-Platform Communication**:

RESTful APIs: As we have been utilizing RESTful APIs in the early development of the application, they will provide a standard and platform-independent means to connect users.

**Network Connectivity and Outages**:

Most importantly, local caching will enable our users to resume a game if a temporary outage occurs. This would prevent the majority of disruptions due to outages. However, major outages may be difficult to alleviate, one consideration would be the implementation of a local version of the game that would allow users to play locally.

**Security**:

While security may not be the most important aspect of ensuring smooth sailing for our client, they have expressed a desire for increased security. Considering this, I suggest the implementation of the following security measures.

**Encryption**:

Ensure that data being transmitted between users and servers are encrypted, ideally HTTPS to protect this data.

**Authentication and Authorization**:

Including but not limited to secure restrictions on account credential instantiation, 2 factor authentication, and a well developed ‘Need-to-Know’ structure for data to prevent unintended exploitation of our systems. Implementation of an RBAC system would be ideal.

**Credential storage**:

Passwords should be stored securely using powerful hashing algorithms, encryptions, and salting techniques.

**Data storage**:

Similar to password storage, data encryption ‘at rest’ should be considered. This can be accomplished through encryption mechanisms provided by Linux.

**Unified Security Policies**:

While not a developmental factor, the enforcement of security policies should be strict across all platforms and devices. A living document of a security design should assist this endeavor.

**Security Maintenance**:

If our client deems it beneficial, it would be advantageous to employ a long-term health inspection plan to keep the security of our client’s application from suffering from new or unrealized vulnerabilities. Such a plan would involve extensive penetration testing of the application, and physical security inspections for on-site insurance.

References:

1.

“*Linux Architecture – Detailed Explanation*.” June 10th, 2022, InterviewBit.

<https://www.interviewbit.com/blog/linux-architecture/>

2.

Narang, Mohita “Kernel and its Types: Explanation with Real-life Analogy.” June 29th, 2023, Shiksha.

<https://www.shiksha.com/online-courses/articles/kernel-and-its-types-operating-system/>

3.

“*Memory Management*.” Date unknown, the Kernel Development Community.

<https://docs.kernel.org/admin-guide/mm/index.html#:~:text=Linux%20memory%20management%20is%20a,and%20in%20man%205%20proc>.

4. (No direct quotation, was a learning resource for understanding concepts.)

Victor, “*Storage Management In Linux Explained With Examples.*” June 23rd, 2020, Tekneed.

<https://tekneed.com/storage-management-in-linux-explained-with-examples/>