

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## 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 | 05/24/2023 | Tomas Estanislao | 1st publish, initial report. Added information up to Domain Model. |
| 2.0 | 06/11/2023 | Tomas Estanislao | 2nd publish, added information up to Evaluation Table. |
| 3.0 | 06/25/2023 | Tomas Estanislao | 3rd publish, added information up to Recommendation section. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room aims to develop a web-based game accessible on various platforms, including Windows, macOS, Linux, Android, and iOS. Their current game, Draw It or Lose It, is limited to Android devices. The application should be scalable; It will support multiple teams and players, unique names, and a single active instance of the game engine. It will render images as clues, with multiple timed rounds and guesses, allowing each team to solve the puzzle. CTS will initiate the project by developing the backend in Java.

## Requirements

Compatibility and scalability are primarily business constraints, while support and singleton are technical—however, they all have a relation to each other.

## [Design Constraints](#_2et92p0)

• Compatibility across multiple platforms: Considerations need to be made in terms of the technologies used on the frontend and backend to ensure the app is compatible on multiple platforms. Using frameworks to build a RESTful web app, from client-side JavaScript to a Java backend, could be a possible solution. Multiple interfaces per device dimension may be considered.

• Scalability: Due to the anticipated growth in demand, the app needs to be scalable to handle increasing user traffic. Big-O time complexity becomes a factor in terms of quickly accessing data and potentially modifying the underlying data structures for the Games, Teams, and Players. Other techniques such as load balancing and caching may be considered.

• Support for multiple teams and players, and unique names: Uniqueness is needed to prevent conflicts and ensure proper synchronization. In addition to enforced data validation, static IDs provide a unique identifier regardless of the location within the system. With a client-server model, the single server can listen for incoming http requests from multiple clients. Concurrent interactions between multiple teams and players, within game instances, may require multithreading.

• Singleton instance for the game engine: The singleton pattern is necessary to optimize memory usage; it avoids duplicate resources and ensures a consistent game state shared between clients.

## [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 ProgramDriver class runs the program with the main() method, and uses the SingletonTester class's testSingleton() method, to verify that the singleton pattern applies to the GameService class. These classes are separate.

The Entity class serves as a base class positioned above the other four sub-classes. It acts as a superclass and holds common attributes and behaviors for the subclasses: Game, Team, and Player. This approach, known as generalization, demonstrates inheritance. It is beneficial in The Gaming Room Java application as it promotes code reuse and provides a way to enforce restrictions. The Entity class has a private constructor that enforces data validation by requiring an ID and name as properties, regardless of the subclass constructor used. Public getter methods are shared amongst subclasses, but there are no setters, resulting in immutable objects. The association relationships connect GameService to Game, Game to Team, and Team to Player, indicating that there can be zero to many instances of the latter associated with the former. Each Game and Team instance maintains a private list of teams and players, respectively, to track associated instances. The Game and Team classes have methods to add multiple teams and players, respectively. Name uniqueness checked using an Iterator before adding to the list, and each subclass has a toString() method that provides a string representation of the object. The static next\_Id variables ensure unique long identifiers for each new instance of Game, Team, and Player created. Lastly, public getter and setter methods increment the static IDs by 1. All fields in all classes are private and can only be accessed through public getter methods, demonstrating encapsulation.

GameService utilizes the singleton design pattern. Private static games contains a list of active games, and private static service allows only 1 service to exist. Private GameService() prevents outside instantiation, and public static getInstance() returns the new or existing service. There are public getter methods to add, get, and count games.

**"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** | - Bash is the default shell, with Zsh and other options available. Homebrew is the package manager for macOS.  - Apple Hardware and licenses are required to use this OS. Server features are transitioning directly into macOS, rather than a separate OS.  - Apache or Docker are two options for deploying this application.  - High initial cost, minimal experience required. Small share of the market hosts on macOS, thus there may be cheaper options available with a large userbase to support troubleshooting and testing. | - Bash is the default shell, with Zsh and other options available.  - Various package managers are default for Linux distros and may include APT, YUM, or Pacman.  - Most distros are free to use with no licensing fee.  - Apache, Nginx, or Docker are options for deploying this application.  - Documentation is widely available for troubleshooting. Security and Stability is rated highly.  - Low cost, high technical experience required. | - Windows PowerShell is the default shell, with Windows Package Manager(winget) included. A GUI may be appreciated by technical staff.  - It is proprietary, with licenses required, and associated fees in the long run.  - IIS or Docker are two options for deploying this application.  - Moderate cost, moderate experience required. Easy to use and install, with long term compatibility. | - Mobile Devices such as an iPhone, Pixel, iPad, and Galaxy Tab use a mobile operating system: such as Android, iOS, and iPadOS.  - They implement mobile UI’s for the user to interact with, and are single user systems.  - CPU, Ram, and Hard drive space would be limiting factors on hosting ability.  - While it may be possible to host a static website from a mobile device, it is not practical nor feasible to serve thousands of users concurrently. |
| **Client Side** | - The app needs to be compatible with Safari, in addition to popular browsers.  - If developing native macOS apps, using Swift or Objective-C would be preferable. This would result in a higher hardware cost and time required, due to the client’s current focus being Android development.  - For all platforms, native testing across devices will be needed to ensure compatibility. Extensive testing of the frontend, JavaScript, is needed to ensure compatibility across systems and browser versions. This web-based app will cut down on cost and team size by having shared codebases and modular features. This allows for rapid Agile development for each browser and OS. | - The app needs to be compatible with Firefox in addition to popular browsers.  - If developing native Linux apps, C/C++, Java, or Python are options. Costs are low due to largely free and opensource software. Much time is required for the initial setup, as well as maintenance and bug fixing. | - The app needs to be compatible with Edge, in addition to popular browsers.  - If developing native windows apps, using C# may be preferable. This would result in a moderate software cost, due to licensing fees. Experience will be necessary in utilizing Microsoft Technologies. | - The app needs to be compatible with Safari and Chrome, in addition to popular browsers.  - Responsive web design is necessary to support smaller client dimensions.  - If developing native Android and iOS apps, using Flutter or React Native could be a cost saving solution. This would result in a low cost, with the client already having an Android app.  - There is also the potential to use these languages to develop a web-app compatible with multiple web-browsers. |
| **Development Tools** | - HTML, CSS, and JavaScript are used on the frontend. It is supported by all major web browsers.  - Node.js or Java can be used on backend.  - Various other languages have cross-platform support and are options.  - Visual Studio Code and XCode are potential tools for macOS. Both are free.  - Version control, such as Git, is standard industry wide.  - Options are very limited for macOS hosting, as majority of the current market share is Linux and Windows hosted.  - Authentication options are available on all platforms. Some may be built-in with the framework, or as part of a third-party service. OpenID and OAuth are common. | - HTML, CSS, and JavaScript are used on the frontend. It is supported by all major web browsers.  - Node.js or Java can be used on backend.  - Various other languages have cross-platform support and are options, such as python with frameworks such as Flask or Django.  - Vim, Emacs, PyCharm and IntelliJ IDEA are popular with developers. There are education and community versions for free.  - Version control, such as Git, is standard industry wide.  - Amazon Web Services and Google Cloud are options for hosting, however there are many affordable options such as: Heroku, Digital Ocean, Vercel, and Linode. | - HTML, CSS, and JavaScript are used on the frontend. It is supported by all major web browsers.  - Node.js or Java can be used on backend.  - C# is a popular option with the ASP.NET framework.  - Visual Studio IDE is a staple among windows developers. Notepad++ and Sublime Text are also used. There are free versions.  - Version control, such as Git, is standard industry wide.    - Microsoft Azure is a cloud hosting option that ties into the existing ecosystem.  - A SQL style database can be implemented if necessary to track and control user information. | - Android Studio is the free official IDE for Android development.  - XCode is Apple's free IDE for both macOS and iOS applications.  - There are various languages and frameworks for Android, iOS, and Cross-Platform mobile apps.  - Android: Java, Kotlin, C++, C#, Python, JavaScript, and Dart.  - iOS: Swift, C#, Objective-C, Flutter, and React Native.  - Cross-Platform: React Native, Flutter, Xamarin, Codename One, Kivy, and RubyMotion.  - Cross-Platform will aid in reducing costs, as well assist with code reusability. Developers may use the same codebase. |

## 

## 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**:

I am recommending Linux as the chosen operating platform. A Linux server is the preferred choice due to stability, security, flexibility, and low costs([redhat.com](https://www.redhat.com/en/topics/linux/linux-server)). An Ubuntu distribution would be the preferred choice, however other distributions may be considered depending on compatibility with web-hosting platforms and other services.

1. **Operating Systems Architectures**:

Linux is structured in a monolithic kernel architecture, wherein the entire system runs in kernel mode([baeldung.com](https://www.baeldung.com/linux/monolithic-kernel#:~:text=Linux%2C%20an%20open%2Dsource%20operating,such%20as%20BSD%20and%20Solaris.)). All system services of this operating system are contained in a single executable and can directly interact with each other. Examples include, memory management, device drivers, file system management, and process management. This makes Linux a fast and responsive system, with less performance overhead and easier maintenance. The specific components of the overall architecture are, “the Kernel, System Library, Hardware layer, System, and Shell utility” ([javapoint.com](https://www.javatpoint.com/architecture-of-linux)). Everything between the Application and Hardware layer are included in the kernel space.

1. **Storage Management**:

A cloud hosting provider such as [DigitalOcean](https://www.digitalocean.com/products/droplets) could be an affordable and scalable storage/hosting solution that has a low initial cost. There are various droplet optimizations available corresponding the application’s needs. They also offer a service to deploy and scale apps for you, without having to manage it yourself. Ubuntu is available as a Linux Distribution with this droplet, as well as pre-installed tools such as: NodeJS, Docker, MongoDB, and MySQL.

Using a CDN to store and deliver customized images closer to the users is recommended. [KeyCDN](https://www.keycdn.com/image-processing) offers image processing to transform and optimize images for each user’s specific browser and device. This cuts down on extra variations of images stored by the company, and caches new images on the edge servers to increase delivery speed and cut processing costs. This is a preferred solution due to the high number and quality of the stock images needed, as well as the cross-platform compatibility requested by the client. It is also important to note that data centers are located globally, allowing for closer image delivery to a global customer base.

1. **Memory Management**:

KeyCDN transforms, optimizes, and caches images for delivery. This allows for a fast and responsive distribution of images to the client. Additional practices such as Lazy loading can assist with the management of client-side memory. “Lazy loading is the practice of delaying load or initialization of resources or objects until they’re actually needed to improve performance and save system resources” ([imperva.com](https://www.imperva.com/learn/performance/lazy-loading/)). The combination of these two tools can provide fast data at the appropriate time, conserving bandwidth. Local and Session storage stored in the browser could also be helpful in keeping repeatably needed information.

1. **Distributed Systems and Networks**:

Hosting the server on a DigitalOcean droplet provides scalability and reliability at an affordable price. There are resizing options for “CPU and RAM only,” as well as “Disk, CPU, and RAM” ([docs.digitalocean.com](https://docs.digitalocean.com/products/droplets/how-to/resize/)). Data is also secured remotely on hardware separate from the droplets to reduce permanent data loss. Their load balancer allows users to connect multiple droplets across a network to distribute the overall user traffic to the Web App. Docker is compatible with DigitalOcean’s droplets, which speeds up development with virtualization. You can perform end-to-end application testing by isolating Docker containers with different environments, in conjunction with a library such as Playwright ([browserstack.com](https://www.browserstack.com/guide/playwright-docker)). This ensures that users across supported browsers and devices will have a seamless and responsive UI.

By implementing a RESTful API with this application, communication between client(s) and server takes place over HTTP ([educative.io](https://www.educative.io/courses/web-application-software-architecture-101/qADAzX6yorR)). Each browser uses various request methods to communicate with the backend server, which is listening to a port, and expects some kind of response through a format like JSON. Since different clients use the same exposed endpoints to request information— yet the functionality of the backend does not change— the clients are decoupled from the backend. This is a client-server architecture that RESTful API facilitates communication between.

1. **Security**:

RESTful APIs are stateless, requiring authentication credentials with every HTTP request. SSL certificates, Basic Auth, OAuth, Password Hashing, API Keys, Timestamps, and Input Validation are all parts of the security of the application ([restfulapi.net](https://restfulapi.net/security-essentials/)). There are frameworks available to assist with these design constraints that assists developers in implementing these checks. Linux known for being secure— in terms of malware, spyware, etc.— administrator privileges are required to make a file executable ([renewablepcs.wordpress.com](https://renewablepcs.wordpress.com/about-linux/advantages-of-using-linux/)). Additionally, due to its open-source nature, countless people are on the lookout for security issues and can share their concerns with established communities. Software updates are much more frequent in comparison to other OS’s. Both KeyCDN and DigitalOcean have security measures in place for their clients such as SSH keys, Cloud Firewalls, and DDOS Attack Prevention.