

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 | 03/19/2023 | Hannah Hendrix | Reviewed and included domain model |
| 2.0 | 03/31/2023 | Hannah Hendrix | Evaluate various platforms for hosting a web-based software application |
| 3.0 | 4/16/2023 | Hannah Hendrix | Included recommendations |

**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 has been hired by The Gaming Room to develop a web-based game, based on their current game named Draw It or Lose It, that can be run on multiple different platforms. Draw It or Lose It is currently only available for Android Operating Systems.

Creative Technology Solutions requires that this game be a multiplayer application with multiple teams, including multiple players per team. There can be only one instance of the game within the memory that exists at any given time. All games and team names require unique names.

In order to confirm that all game and team names are unique, the software must not allow multiple instances of the same name to be allowed, while also notifying the player that the name is available or unavailable. This can be accomplished using an iterator pattern.

A singleton creation pattern has been developed for object creation to prevent multiple game instances within the game memory.

## Requirements

Creative Technology Solutions has expressed that they require the ability to have multiple different teams (one or more) that can include multiple different players assigned to said teams.

There can be only one instance of a game that exists in the memory at any given time, while including unique identifiers for game, team, and player names.

## [Design Constraints](#_2et92p0)

Draw It or Lose it currently features an Android-based deployment that The Game Room already uses. Since Creative Technology Solutions has been asked to expand Draw It or Lose its availability to the web, the development of this project must include web-deployment. Web-deployment requires extensive knowledge of Java, therefore for this expedition the developmental team must have high function skills in Java. The team needs to be fluent in Android operating systems as the existing APIs serving the platform must be revised.

Since Java is the native Android SDK language, this can ease the transition and reduce costs for needed developers in many different languages.

## [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)

A main driver class in the program is what will be used to begin producing the games, teams, and players. Only one GameService class may ever exist in memory at any given time because to the singleton design pattern used in the GameService class, which performs the actual construction.

By changing its constructor to private, GameService prevents any instances from being created. The getInstance() method is the sole way to create a GameService. getInstance() checks whether GameService has been launched and only starts if it is not present in memory.

The driver class can call the function addGame once GameService is active (). The iterator pattern is used by addGame() to stop the creation of Game objects with the same name. The List of games is then updated to include this new Game object.

The addTeam() method can be used to add a team to the game once it has been created. The iterator pattern is used by addTeam() to stop the addition of Team objects with the same name to the game. The List teams then includes this new Team object.

With the addPlayer() method, a player can be added to the team after it has been created. The iterator pattern is used in addPlayer() to stop other Player objects with similar names from being added to the team. The List of players then includes this new Player object.

Subclasses of Entity include the classes for Games, Teams, and Players. Id and name are two of the entity's protected characteristics. Only the overloaded constructors may be used because the default constructor is protected and prevents the generation of null objects.

Several Object-oriented programming strategies are displayed in the developed UML. The overloading of constructors and the extension of the Entity class both make use of polymorphism and inheritance. The methods used to add teams are also characterized by encapsulation and abstraction. As access to the constructor is restricted, a Team object cannot be formed directly, but the addTeam() method still permits creation of a Team object without the user being aware of how the team was added.

The UML diagram displaying the functions of the design is provided below:

**"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** | A server can be obtained at a modest cost of $20 (as stated on the Apple website) in order to operate IOS services and develop websites. In addition, their website presents abundant resources for macOS servers to initiate the usage of the server. Nevertheless, comparatively few individuals prefer Apple as their primary operating system. Apple allows for flexible terminal commands in order to configure the server. It has more software available than other operating systems. However, it is a closed platform. It takes less time to load, is more comfortable to use, and does not require many external resources, but it is susceptible to viruses. | There is no cost for the Linux license, but the price is determined by the server’s requirements and what is able to be run on it, this will require a team focused solely on research and cost estimation. A disadvantage of Linux is that few people are familiar with the interface. Also allows for flexible terminal commands in order to configure the server. It has a variety of web hosting options to meet a variety of needs. The operating system can also be upgraded in response to market changes, but it is less popular for web hosting services. | Unlike Linus and Apple, Windows requires a much higher license fee. They have several options starting at $500. From 2022, the standard price is $1069, and the Data Center Edition is $6,155. Both of these prices are for owning and using. Despite being the most expensive option, Windows OS is the most widely used operating system today. Windows has the ability to run on more software than other systems. It is extremely safe as Windows detects security threats before they become problems. Windows has the ability to accommodate the large number of people. | Mobile Servers can range anywhere from $70 up to $320 per month. This is determined by the amount of CPU, memory, and disk space available. It's reasonably priced, but it depends on the scope of the project. However, if the app receives a lot of traffic, the costs will be easily covered. And the vast majority of people use mobile devices to access data. A stationary server that can be tracked to a specific location. Because the specifications are better, it would be one of the top contenders. Because of its shared nature, the platform is less expensive, more compatible, and has a greater reach. It does, however, have poor security management and is limited to specific smartphones. |
| **Client Side** | Regardless of the operating system, development should ensure optimized, modular processes that handle a large number of requests while returning quick, lightweight responses to the client. This requirement is met by the REST API structure. Modern browsers, secure authentication, and transmission protocols that prevent session hijacking are all supported. Mac OS only supports a few file structures, which can limit media types. | Only a few file structures are supported. Because out-of-the-box file type support is limited and requires additional programming to handle additional types, this limits the media type options that can be delivered to the client without additional programming overhead. | Optimized for use with Microsoft's internally developed web browsers, which may pose a technical limitation. However, all major web browsers are supported, and the REST structure is supported by Microsoft web browsers. | Due to insufficient processing power and a lack of plugin architecture to support heavyweight or executable media types, all logic must reside on the server. Ideal for lightweight REST APIs with clear and precise message exchange. Expertise in developing for optimal user experience across multiple form factors is required. |
| **Development Tools** | This architecture enables multi-threaded one-to-one capabilities, allowing for scalability, concurrent, and fast development that are not dependent on the operating system. Complied languages can make use of a great variety of libraries and the object-oriented programming paradigm to facilitate deployment on all major platforms, while open source frameworks and integrated development environments like Maven and Eclipse provide cost-free and OS-independent solutions. | Linus allows for a multi-threaded, one-ton-one approach that adds extra overhead. Compiled languages like Java, can reduce this effort while interpreted languages come with additional runtime effort. Interpreted languages such as Python are not genuine object-oriented-oriented programming solution and can impede scalability or restrict design possibilities. | Threading model is one-to-one. The overhead introduced by compiled languages (such as C#) is offset. Windows is optimized for its own.NET framework, which uses C# for object-oriented programming. Other languages are also supported, limiting the scope of this technical limitation. | Different programming languages are preferred by iOS and Android. Android prefers Java, which iOS does not support natively. To work across platforms, a framework or platform-specific language may be required. Cross-platform development necessitates knowledge of specific frameworks or multilingual programming practices. |

## 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 client Gaming Room ought to utilize a cloud-based or server-less architecture with a Linux-based operating system. Serverless solutions offload maintenance, acquisition, and ownership costs to a third party that can offer competitive pricing due to scale. Through single-purpose autonomous services and frameworks, it reduces additional complexity and errors. It encourages scalable reliable methods with lower maintenance and provisioning costs. These services can be accessed via a variety of methods, including HTTP, API, and others, allowing the product to better support distributed operations across multiple and varying computing environments. The serverless pricing model encourages efficient design by relying on fewer dependencies to reduce processing time. Middle-layer service solutions can reduce the scope and complexity of the software's traditional logic layer substantially. Certain features can be utilized from within the client's interface, which reduces delay and encourages simpler application architecture. These factors result in a faster-running application with better user interfaces and an established solution that allows for faster enhancements in execution.
2. **Operating Systems Architectures**: A middle layer that performs logic and transmits data between the user view and the information store is incorporated in conventional three-tier methods. Serverless architectures are built on this framework by structuring this component. Many middle-layer solutions, such as security services, the transmission of content, and business logic operations, may be utilized to deliver an effective application. This structuring, which is comparable to a microservice framework on a bigger scope, enables the client to consume and purchase only the services that it demands. The Linux operating system is built in a modular format. The operating system's kernel contains exclusively fundamental components, while independent modules allow for flexible service development. This framework results in an efficient approach that takes up fewer resources on the disk. Modular operating systems also avoid the more latent means of communication implemented in layered operating systems. This increases productivity and processing speed for an improved experience for users. Several operations in Linux are also carried out by system libraries, which have no direct access to the kernel. This integrates security by restricting the utilization of device hardware.
3. **Storage Management**: In the storage management layer, serverless solutions utilize multiple servers as well as distributed technology to produce various mediums of storage under a single user interface. This layer incorporates storage capacities into a resource pool, promoting increased redundancy and cloud computing space. When utilized alongside indexed, direct-access storage procedures cloud computing enables the creation of programs without regard for the amount of memory available. Direct access storage provides the quickest execution times, the best cloud computing assistance, and a superior user experience. Indexing allocation strategies reduce the demand for unnecessary input and output sequential data reads that conventional distribution methods demand while additionally eradicating wasted storage space due to disintegration triggered by the contiguous allocation of resources. Other allocation strategies produce difficulties with performance and may decrease uptime by eliminating concerns about fragmentation. Multi-tiered indexed allocation is used in the Linux directory framework. Each file or directory has its own identifier. To save memory space by avoiding large, single indexes, these identifiers are utilized throughout several index layers. To reduce seek time, Linux stores files on blocks near their index. These elements minimize the effective memory access time.
4. **Memory Management**: While care ought to be utilized in order to prevent incorporating unneeded code to the application, the source code will include additional components compared to what is executed in one operation. The CPU has to execute the application that exists just partially in storage. Linux makes use of virtual storage and on-demand paging to perform partial memory execution. This removes physical memory size limitations and enables numerous applications to run concurrently. Partial memory execution also speeds up the loading and running of the application, which improves user experience. Page faults are produced by demand paging. If these errors aren't dealt with properly, they have a negative effect on performance. Operations are stored on the hard drive and are divided into smaller logical categories that are only used in physical memory for execution if necessary. Page databases are used to organize places in memory and monitor pages that are currently in memory. They are located between the virtual and physical spaces. Every application process in Linux is assigned to a table.
5. **Distributed Systems and Networks**: Higher client engagement characterizes applications that are effective. The serverless architecture scales effortlessly to accommodate advancement. To accommodate demand from clients, the application must be capable of scaling throughout numerous servers as it develops. Distribution of load and reproduction are critical for successfully supporting elevated usage by clients across multiple systems. A load-balancing system allocates client demands throughout application servers, increasing reactivity and accessibility. In order to accommodate numerous clients, the application should provide an API that serves every client in a consistent manner. Because APIs are portable, this encourages cross-platform support. Client-server REST API exchanges over HTTP are compact and stateless, with the ability to support distinctively identifying session data. Communications are consistent and originate from the client. Every communication method type reflects the same context to all participants, ignoring client differences. Messages are centered on particular assets that perform specific tasks. Solutions built with serverless architecture are built on top of individual functions or parts of business logic. REST materials are easier to associate with these functions than other API structures. The internet is used for interaction among clients and the application's server. The protocol known as HTTP is used in Internet communication. Since HTTP is stateless, developers have to operate within constraints in order to create a stateful experience. This endeavor is aided by particular session identifiers. The development of these identifiers, as well as their management, must guarantee that the proposed solution is not dependent on excessive amounts to have an adverse effect on productivity or grow challenging to maintain. Several load balancers also offer productive and low-cost protection. Such safety precautions can supplement firewall protection, which has been lacking as of late. Network administrators may utilize one of many load-balancing for load balancing to guarantee ideal server use and the fastest possible response times for clients. In a distributed system, the application's database is typically run on several computers at the same time. Database instances must be constantly synchronized in order to appear as just one instance. By establishing a single write node that generates numerous read-only nodes, the master-slave reproduction method utilizes continuous synchronization. This strategy takes advantage of the fact that incorporating data into the database is much less common than obtaining data. When data on the central instance is modified, slave nodes are notified that the change is now available for recovery. Via redundancy, a distributed system that includes load balancing successfully addresses problems with connectivity. If the application's server instance is down, the load balancer can redirect traffic to other servers, preventing interruptions in operation. Because beneficial database instances remain operational to serve the application even when other instances are down, a distributed database solution provides comparable functionality to the application. Distributed systems encourage continuous operation with no downtime. Fresh code can be implemented sequentially to pools of application servers while other application servers continue to serve users.
6. **Security**: Via process independence, Linux encourages privacy and security. Every operation in the operating system has a distinct virtual address space that is separate from all other processes. This hinders one application's process from impacting another, in addition to safeguarding specific memory regions from being overridden by independent programs. Furthermore, each Linux user must authenticate in order to use the operating system, and least privilege is imposed automatically. This restricts access and possible threats. Linux uses discretionary access control, or DAC, to secure files and directories. With DAC, file owners can specify which individuals or groups should have access to access and modify the file. Access control lists are used to enforce DAC. Authorization along with authentication are critical components of developing secure software applications throughout multiple platforms. For this application, login information is the most suitable authentication approach. Alternative approaches, such as biometrics provide overly complex solutions which could limit accessibility for users or negatively impact user experience. User authorization should be conducted using role-based access control, or RBAC, that encourages the principle of least privilege. Users ought to be assigned player roles which enable them to participate in games and supervise teams while not modifying the application in any way. Administrators can be assigned roles which provide additional accessibility, such as handling library files, modifying puzzles, or changing application configurations. REST-based lightweight communication among clients and servers encourages stateless interaction, which necessitates the use of cached client-side identifiers to replicate a stateful experience. The unique identifiers for this application are likely to be session-specific and do not call for persistent client-side identifying elements such as cookies. Persistent confidential information needs to be stored in the database in a hashed state, preventing a plaintext view and providing additional security if the database is accessed in an unauthorized manner. Client-side web browser security and transmission encryption are critical to preventing session hijacking by ensuring that identifying session data is not intercepted. The encryption techniques used should be up to date and incapable of being reverse-engineered in order to decode the information being transmitted.