

Draw-it or Lose-it

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/10/22 | Wesley Blackwell | Assessing the design and business constraints of the customer’s online gaming application |

**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)

Based on their current game, Draw It or Lose It, The Gaming Room would like to develop a multi-platform and multi-team, web-based game application with unique game, team, and player names as well as validation for uniqueness. Using polymorphism, we will create a class system with “Entity” as the parent class and “Game”, “Team”, and “Player” will be its subclasses. From that framework, we can add multiple teams, assign multiple players to teams, and apply Singleton and Iterator patterns to perform validation to ensure only one game instance exists in memory at any given time.

## [Design Constraints](#_2et92p0)

**Functional Requirements**

1. Each game will have more than one participating team
2. Each team will be comprised of multiple players
3. Each team, game, and player will have unique names
4. The game will only have one instance in memory at a time

## [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 program driver class contains main() and uses the singleton tester class. The entity class is the parent of the three child classes; game, team, and player. From right to left, the player is associated with the team class. The team class is associated with both the player and game classes and can contain zero to many team members. The game class is associated with the game service and team classes and can contain zero to many teams. The game service class is associated with the game class and can contain zero to many games but only one service which means only one game can be active in memory at a time.

**Abstraction**

The team class contains the private list of player type. The game class contains the private teams list of team type. The game service contains the private list of games of game type.

**Encapsulation**

We are using classes containing specific private variables. We are also using public and private methods

**Inheritance**

Game, team, and player classes inherit attributes from the entity class.

**Polymorphism**

There are methods with the same name but require different data types as input.

**"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** | Mac servers are the most expensive. Features include: Caching Server, File Sharing Center, Xcode, and Time Machine Server. The current price for a license is 129$. | Linux offers high stability and flexibility but is less compatible for gaming support, and low driver creation support and flexibility. The OS can be obtained for free. | Windows has a UI Desktop that is easy to use. Features: Admin Center, HTTP/2, App Compatibility, Software Defined Networking. Standard Server OS price: $577.99 | Android and iPhone are both user friendly and well supported. Android has a higher security risk because the app store is not strictly monitored like iPhone. Cloud based |
| **Client Side** | Mac multitasks efficiently but is expensive and is difficult to integrate with non-Apple products. The OS is easy to use and convenient. | Linux is not as user friendly as Windows and Mac. It is the most cost-effective option. | Windows is expensive. The OS has low security so there will be additional work to do in that department. It is also low maintenance so there will be some savings. | Android’s low security will add some time to the project. iPhone will have to test and approve the application before it can be on the app store. |
| **Development Tools** | * Xcode IDE * Firefox for testing * MAMP * Apache * PHP | * Bluefish IDE * Firefox * VS Code IDE * Pinegrow | * Visual Studio * Eclipse * Bootstrap * Maven * Jersey | * Xcode – iPhone * Android Studio * Xamarin – multi platform |

## 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 Gaming Room should use cloud-based architecture using the Linux operating system. This will allow them to contract maintenance, procurement, and ownership costs to a third party for a recurring fee. It reduces complexity and inefficiencies through single-purpose autonomous service. It also, promotes scalable and efficient solutions with low maintenance and provision overhead. There are many ways to access these services (HTTP, API, and other services). This allows the solution to better support distributed operations across varying platforms and environments. Utilizing cloud servers reduces the complexity of the overall system creating fewer dependencies. Applying middleware to the system will reduce the scope and complexity of the application’s logic layer. Some functionality (e.g. user authentication, file uploads, etc.) can be accessed directly from the client, reducing latency and promoting more simplistic app architecture (Sbarski, n.d.). These factors result in a faster application and enable quicker implementation of improvements to the product.
2. **Operating Systems Architectures**: Three tiered solutions use middleware to pass information between the user and data storage. Serverless systems build on this through modularization of the middle layer. Middleware can be used to develop a robust application with security, content delivery, and logic for business. These allow The Gaming Room to utilize the service and only pay for the required services.

The Linux operating system is constructed in a modular fashion as well. The kernel only contains core components while other modules promote dynamic expansion of services. The design of this structure is lightweight and allows less storage use. Linux executes many functions through system libraries that cannot access the kernel directly. This provides extra security through limiting access to pieces of the hardware system.

1. **Storage Management**: Serverless solutions leverage multiple servers and distributed technology to synthesize varied storage media under a common interface in the storage management layer (Zehua, n.d.). This layer integrates storage spaces into a resource pool that promotes greater redundancy and virtualization space (Zehua, n.d.). Virtualization allows program design without consideration for memory capacity (Silberschatz, 2009) and is most efficient when combined with indexed, direct access storage methods. Direct access storage offers the fastest load times, best virtualization support, and best UX. Indexed allocation schemes eliminate the need for unnecessary input/output sequential data reads required with other allocation methods (Silberschatz, 2009). Indexed allocation also eliminates wasted storage space caused by fragmentation stemming from contiguous allocation. Removing fragmentation issues under other allocation schemes yields performance issues and can reduce uptime (Silberschatz, 2009).

Linux has a file system structure that utilizes multi-tiered indexed allocation. Each file or directory has a unique identifier. These identifiers are referenced across multiple index tiers to conserve memory space by avoiding large, single indexes (Silberschatz, 2009). Linux looks to store files on blocks near their index to reduce seek time (i.e. the amount of time needed to find the file on disk once the reference is known) (Silberschatz, 2009). These factors reduce memory effective access time (i.e. the time needed to bring processes in data into memory for execution) (Silberschatz, 2009).

1. **Memory Management**: The code base will contain more items than will run in one process. Take care not to add useless code in the application and to remove old code whenever necessary. Utilizing Linux’s partial memory execution feature, the CPU can execute the application so it only partially exists in memory. This allows the application to load quickly and run fast to improve user experience. Through virtual memory and demand paging, the physical memory size constraints can be eliminated, and multiple programs can run in parallel.

Demand paging generates page faults. These faults impact performance when not managed effectively (Silberschatz, 2009). Processes reside on disk and are broken into smaller logical segments (pages) that are only pulled into physical memory when needed for execution (Silberschatz, 2009). Page tables reside between the virtual and physical spaces to map memory locations and track pages presently in memory (Silberschatz, 2009). Linux uses a table for each app process (Rusling, 1999).

The swap algorithm influences performance and the level of thrashing (Rusling, 1999). With high thrashing, page exchanges are not optimally performed and too much effort is spent swapping between physical and virtual space (Rusling, 1999). This adversely affects performance and UX. Linux employs the Least Recently Used (LRU) algorithm to manage page swaps (Rusling, 1999). Older, relatively dormant, pages become candidates for swapping under this approach (Rusling, 1999). This keeps the thrashing low and keeps up performance levels for a good user experience.

1. **Distributed Systems and Networks**:

Serverless architecture allows for seamless scaling to support growth. Load balancing and replication support increased client use across the distributed system as the application scales to handle client demand. Load balancing evenly distributes the client requests across the application servers. Many load balancers also provide added security. These security measures can bolster firewall protection that has proven limited in recent years (What is Load…, n.d.). Network administrators can implement one of several load balancing algorithms (e.g. least connection method, round robin, etc.) to ensure optimal server use that provides clients with the fastest response times (What is Load…, n.d.).

In a distributed system, the app database is usually run across several machines simultaneously (Kozlovski, 2018). Database instances must be synced continuously to appear as a single instance (Kozlovski, 2018). The master-slave replication strategy uses continual synchronization by implementing a single write node (i.e. the master database) that generates several read-only nodes (i.e. slave databases). This strategy leverages the idea that adding data to the database (i.e. writes or edits) is much less frequent that retrieving data (i.e. reads) (Kozlovski, 2018). When data is modified on the master instance, it informs slave nodes the change is available for subsequent retrieval (Kozlovski, 2018).

Clients and servers connect via the internet. HTTP is used in internet communication. Unique session identifiers aid developers in working in HTTP’s stateless nature constraints. The development and management of the identifiers should ensure the solution does not have too many of them as to bog performance or maintenance.

The API used should service all clients to promote multi-platform use. REST API exchanges between clients and servers over HTTP are lightweight and stateless (What is REST…, n.d.). These exchanges are also capable of supporting unique identification data. Uniform messages are initiated by the client. Each message method type (i.e. GET, POST, PUT, DELETE) represents the same context to all parties- discounting nuances between clients. Messages center around specific resources (i.e. server-side artifacts identified by a unique uniform resource identifier) that carry out specific functions (What is REST…, n.d.). With serverless architecture, solutions are built on top of individual functions or pieces of business logic (Bashir, 2019). REST resources are more easily mapped to these functions than other APIs.

Distributed systems with load balancing effectively combat connectivity issues through redundancy. If an application server is unavailable, the load balancer can move traffic to good servers. Distributed database solutions serve a similar role to the application. Good database instances can continue to serve the application when other instances are down. Distributed systems also utilize continual deployment with the goal of zero downtime. New code can be deployed multiple application servers in sequence while leaving other application servers up and running.

1. **Security**:

Authentication and authorization are necessary for security over a distributed environment. Login credentials are appropriate for the application. Other methods like biometrics would make using the application too complicated and bring down the user experience. The best way to use authorization is through role-based access control (RBAC) that promotes least privilege. Users should be given privileges to use the application but not allowed to make changes to it. Administrators should be allowed to change configurations, manage libraries, and edit puzzles.

Lightweight communication between clients and the server using REST promotes stateless interaction that requires cached client-side identifiers to simulate a stateful experience (Parikh, 2020). Identifiers are likely to be unique to each session and not require more persistent client-side identifiers. Persistent sensitive data (e.g. user passwords) should be stored in the database in a hashed state that prevents a plaintext view (Silberschatz, 2009). This adds extra security if the database is accessed in an unauthorized way. Client browser security and transmission encryption should be current to prevent session hijacking.

Linux promotes security through process independence. Each process in the system has its own virtual address space independent of other processes (Rusling, 1999). This prevents a process running one application from affecting another while also protecting certain memory areas against being overwritten by rogue applications (Rusling, 1999). Additionally, each Linux user requires authentication to use the OS and least privilege is enforced automatically. This limits scope of access and potential threats (What is Linux…, n.d.). Linux protects files and directories by way of discretionary access control (DAC). With DAC, file owners can dictate other users or groups who should have access to read or edit the file (Overview of Linux…, 2013). DAC is enforced through access control lists (ACLs) (Overview of Linux…, 2013).

**References:**

Bashir, Faizan. (2019, July 5). *What is Serverless Architecture? What are its Pros and Cons?*

Retrieved from <https://www.freecodecamp.org/news/what-is-serverless->

architecture-what-are-its-pros-and-cons/.

Kozlovski, Stanislav. (2018, April 27*). A Thorough Introduction to Distributed Systems.* Retrieved from

<https://www.freecodecamp.org/news/a-thorough-introduction-to-distributed-systems->

3b91562c9b3c/.

*Overview of Linux Kernel Security Features.* (2013, July 11). The Linux Foundation. Retrieved from

https://www.linux.com/training-tutorials/overview-linux-kernel-security-features/.

Parikh, Aarti & Albert Agram. (2020). *Introduction to HTTP.* Launch School.

https://launchschool.com/books/http.

Rusling, David. (1999). *The Linux Kernel.* David A. Rusling. https://www.tldp.org/LDP/tlk/tlk.html.

Sbarski, Peter. (Not Dated.). *The essential guide to serverless technologies and architectures.*

Retrieved from <https://techbeacon.com/enterprise-it/essential-guide-serverless->

technologies-architectures.

Silberschatz, Abraham, Peter Baer Galvin, & Greg Gagne. (2009). *Operating Systems Concepts,*

*Eight Edition.* John Wiley & Sons, Inc. <https://learning.oreilly.com/library/view/>

Operating-system-concepts/9780470128725/.

*What is Load Balancing?* (Not Dated.). AVI Networks. Retrieved from

<https://avinetworks.com/what-is-load-balancing/>.

*What Is Linux and Is It Really Secure?* (Not Dated.). Kaspersky. Retrieved from

https://usa.kaspersky.com/resource-center/definitions/linux.

*What is Rest?* (Not Dated.). RESTfulAPI.net. Retrieved from https://restfulapi.net/.

Zehua, Cai & Wang Shunyan & Long Shangyin & Zhou Haitao. (Not Dated.). *A Data Storage and*

*Management Scheme in Cloud Storage Model.* (Wuhan University of Technology, Wuhan,

China). Retrieved from https://download.atlantis-press.com/article/5031.pdf*.*