

<Draw It or Lose It>

# **CS 230 Project Software Design Template**

Version 2.0

## 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 | May 26, 2024 | Hiep Ha | Executive summary, System architecture view |
| 1.2 | May 30, 2024 | Hiep Ha | Design constraints, domain model, evaluation |
| 1.3 | June 1, 2024 | Hiep Ha | Added recommendations |
| 1.5 | June 5, 2024 | Hiep Ha | Evaluate development needs and OS platform characteristics |
| 1.8 | June 9, 2024 | Hiep Ha | Analyzed techniques and characteristics of system architectures, make a recommendation to The Gaming Room. |
| 2.0 | June 23, 2024 | Hiep Ha | Detailed recommendations for operating platform, OS architectures, storage and memory management, distributed systems, and security to The Gaming Room. |

**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)

CTS has been tasked by The Gaming Room to develop a web-based version of their Android game "Draw It or Lose It." The objective is to enable multiple teams, each with multiple players, while ensuring that each game instance, team, or player occurs only once.

To address this, CTS has adopted a singleton design pattern for object creation, preventing multiple game instances, and will implement an iterator pattern to avoid conflicts between teams and team members.

## [Design Constraints](#_2et92p0)

The existing Android deployment of Draw It or Lose It by The Gaming Room requires compatibility with web deployment. Thus, Java has been chosen as the technology stack for this project. Leveraging Java, which is the native language for Android development, will facilitate the transition to the web platform.

Additionally, any existing APIs serving the Android platform will need to be reviewed or extended to support mobile usage.

## [System Architecture View](#_ilbxbyevv6b6)

Although not immediately relevant to this project, describing the system and subsystem architecture, including physical components or tiers, may be necessary for future projects. A logical topology of communication and storage aspects is crucial for understanding the overall architecture and should be provided as needed.

## [Domain Model](#_8h2ehzxfam4o)

The UML for the proposed design is shown 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.

The application is structured around a main driver class responsible for initiating the creation of games, teams, and players. The creation process is managed by the GameService class, which follows the singleton design pattern, ensuring that only one instance of GameService exists at any given time.

To achieve this, GameService's constructor is set to private, preventing direct instantiation. Instead, instances of GameService are obtained through the getInstance() method, which checks if a GameService instance is already running before starting a new one.

Once GameService is active, the driver class can utilize the addGame() method to create new games. This method employs the iterator pattern to prevent the creation of duplicate games with the same name, adding the new Game object to a list of games.

Teams can be added to games using the addTeam() method, which similarly employs the iterator pattern to prevent duplicate team names within a game. The newly created Team object is then added to a list of teams associated with the game.

Similarly, players are added to teams using the addPlayer() method, which prevents the addition of players with duplicate names to a team. The newly created Player object is added to a list of players associated with the team.

All classes involved in this process, including Game, Team, and Player, inherit from the Entity class, which defines common attributes like id and name. The Entity class's protected constructor ensures that only valid instances can be created using its overloaded constructors.

This design leverages object-oriented principles such as polymorphism, inheritance, encapsulation, and abstraction to provide a flexible and robust system for managing games, teams, and players.

## [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 must 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** | Server-Side Mac offers stability and security for hosting web-based applications but may have scalability limitations and higher hardware costs. | Linux is known for its flexibility and strong community support, making it ideal for servers, although setup and configuration may require more technical expertise. | Windows provides good compatibility with development tools and services but may be more vulnerable to security threats and entail licensing costs | Mobile devices can host web-based applications but may face limitations in processing power, memory, and network connectivity. |
| **Client Side** | Client-Side Mac clients may require specialized knowledge and resources due to tools like Xcode and Swift. | Linux clients need expertise in open-source tools and frameworks, given the varied distributions. | Windows clients offer a standardized environment with support for Visual Studio and .NET but may involve licensing costs. | Mobile devices present unique challenges, needing optimization for performance and usability on smaller screens and touch interfaces, and expertise in mobile-specific languages and frameworks like Swift, Kotlin, SwiftUI, or React Native. |
| **Development Tools** | Development Tools For deploying on Mac, use Java, JavaScript, HTML, CSS, Spring Boot, or Node.js with IDEs like IntelliJ IDEA or Visual Studio Code. | For Linux, utilize C, C++, Python, or web technologies with IDEs like Visual Studio Code or Eclipse. | Windows development involves C#, Visual Basic, .NET, and Visual Studio. | Mobile app development requires Swift, Objective-C, Kotlin, or Java with frameworks like SwiftUI, Android SDK, or React Native, and IDEs like Xcode or Android Studio. Cross-platform tools like Flutter or Xamarin offer multi-platform development. |

## 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 recommend adopting a cloud-based platform such as Amazon Web Services (AWS) or Microsoft Azure. These platforms offer scalability, flexibility, and compatibility across various computing environments, allowing The Gaming Room to expand Draw It or Lose It seamlessly.

**2. Operating Systems Architectures:** AWS and Azure support a wide range of operating systems architectures, including Windows, Linux, and macOS. They provide virtual machines, containers, and serverless computing options, accommodating diverse architectural needs.

* **Virtual Machines (VMs):** VMs provide an environment that emulates a physical computer. This approach allows The Gaming Room to run different operating systems in isolation on a single physical machine. For instance, AWS EC2 instances can run various OS images, providing flexibility in development and deployment.
* **Containers:** Containers, such as those managed by Docker and Kubernetes, offer lightweight, portable, and consistent environments for application deployment. Containers encapsulate the application and its dependencies, making it easier to move between environments and scale applications.
* **Serverless Computing:** Services like AWS Lambda or Azure Functions allow for running code without provisioning or managing servers. This can be particularly useful for scaling certain parts of the game dynamically based on demand.

**3. Storage Management:** For storage management, a cloud-based solution like Amazon S3 (Simple Storage Service) or Azure Blob Storage is suitable. These platforms offer scalable, durable, and secure storage options with features for data encryption, access control, and data redundancy.

* **Amazon S3:** S3 provides high durability, availability, and performance object storage. It supports various storage classes, lifecycle policies for data archiving, and integrates seamlessly with other AWS services.
* **Azure Blob Storage:** Azure Blob Storage offers similar capabilities with hot, cool, and archive access tiers to optimize storage costs. It also provides robust security features and integration with Azure services.

**4. Memory Management:** The recommended operating platforms handle memory management through resource allocation and management tools provided by the cloud service provider. They dynamically allocate memory resources based on application demands, optimizing performance and minimizing overhead.

* **AWS:** AWS offers tools like AWS Elastic Beanstalk, which automatically handles capacity provisioning, load balancing, and auto-scaling. Amazon RDS (Relational Database Service) manages memory allocation for database instances.
* **Azure:** Azure provides similar services, such as Azure App Service for web applications and Azure SQL Database for managed database services. Both platforms offer auto-scaling and performance monitoring tools.

**5. Distributed Systems and Networks:** Draw It or Lose It can communicate between various platforms using distributed software architecture such as microservices or serverless computing. By deploying components as independent services, communication between platforms can be facilitated through APIs or messaging queues.

* **Microservices:** This architectural style structures an application as a collection of loosely coupled services. Each service can be developed, deployed, and scaled independently. Tools like AWS ECS (Elastic Container Service) or Azure Kubernetes Service can manage microservices.
* **Serverless Computing:** Functions-as-a-Service (FaaS) like AWS Lambda or Azure Functions allow for running small pieces of code in response to events, facilitating seamless interaction between services.
* **Network Considerations:** Utilizing services like AWS VPC (Virtual Private Cloud) or Azure Virtual Network ensures secure and efficient network communication. Load balancers, such as AWS ELB or Azure Load Balancer, distribute incoming application traffic across multiple targets, enhancing availability and fault tolerance.

**6. Security:** Security measures should be implemented at multiple levels to protect user information on and between platforms. The recommended operating platforms offer built-in security features such as identity and access management, encryption, and network security groups.

* **Identity and Access Management:** AWS IAM and Azure Active Directory provide robust access control mechanisms, ensuring only authorized users and services have access to resources.
* **Encryption:** Both AWS and Azure offer encryption for data at rest and in transit. Services like AWS KMS (Key Management Service) and Azure Key Vault help manage encryption keys securely.
* **Network Security:** Using security groups, network ACLs, and VPNs ensures secure communication between services and external users. Additionally, implementing Web Application Firewalls (WAF) protects against common web exploits.
* **Best Practices:** Adopting secure coding practices, conducting regular security audits, and ensuring compliance with industry standards (such as GDPR, HIPAA, etc.) are crucial for maintaining a secure environment.

By leveraging the capabilities of AWS or Azure, The Gaming Room can efficiently scale and manage their game, Draw It or Lose It, across various operating platforms, ensuring performance, security, and seamless user experience.