

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 | Jul /20/2023 | Jeury Santos | the client has requested that the following software requirements be met for the game application:  A game will have the ability to have one or more teams involved.  Each team will have multiple players assigned to it.  Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.  Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player. |

**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)

The Gaming Room has expressed the desire to develop a web-based version of their existing Android game, "Draw It or Lose It," which is inspired by the classic TV game show "Win, Lose or Draw." The web-based game will allow multiple teams to compete by guessing stock drawings as clues rendered by the application. Each game will consist of four rounds, and teams will have one minute to guess the puzzle in each round. If the puzzle remains unsolved after a minute, other teams will have a chance to offer one guess within a 15-second time limit.

The application will be designed to support multiple teams, each with multiple players assigned to them. To ensure uniqueness, the game and team names will be enforced as unique identifiers. Additionally, we will implement a mechanism to ensure that only one instance of the game exists in memory at any given time.

## Requirements

*Technical Requirements:*

*Web-Based Application: The game should be developed as a web-based application accessible through web browsers on various platforms.*

*Server-Side Management: The application requires server-side management to handle game sessions, teams, and players.*

*Unique Identifiers: The system should create unique identifiers for each game, team, and player to manage instances effectively.*

*Scalability: The application should be scalable to handle multiple concurrent users and game instances.*

*Data Storage: The game progress, team, and player information need to be stored in a database for persistence.*

*Security: Strong security measures should be implemented to protect user information and prevent unauthorized access.*

*Cross-Platform Compatibility: The application should be compatible with various client devices, including desktops, laptops, and mobile devices.*

*Performance: The application should offer smooth and responsive gameplay with minimal latency.*

*Testing: Thorough testing should be conducted to ensure the application's functionality and identify and address potential issues.*

*Development Tools: Appropriate programming languages, frameworks, and development tools should be used to efficiently build and deploy the application.*

## [Design Constraints](#_2et92p0)

Developing the game application in a web-based distributed environment presents certain design constraints that must be considered during application development. These constraints include:

Platform Compatibility: The web-based game application must be compatible with various operating platforms, including Mac, Linux, and Windows, to cater to a broader audience. Additionally, the application should be optimized for mobile devices to reach a wider user base.

Network Communication: As a web-based application, the game will rely on network communication to facilitate gameplay and interaction between teams and players. Network latency and reliability should be taken into account to ensure smooth gameplay.

Data Security: Given the competitive nature of the game, data security is paramount. Measures should be implemented to protect user information and prevent unauthorized access to game data.

Scalability: The application should be designed with scalability in mind to accommodate potential growth in the user base and increased demand during peak times.

User Experience: A seamless and intuitive user experience is crucial for the success of the game. The interface should be user-friendly and responsive across different devices and screen sizes.

## [System Architecture View](#_ilbxbyevv6b6)

Logical Topology:

The logical topology of the web-based game application would typically involve the following components:

Client-Side:

Web Browsers: The client-side of the application will primarily be accessed through web browsers running on different platforms (e.g., desktops, laptops, mobile devices). The application's user interface will be rendered in the browser, allowing players to interact with the game.

Web Server:

The web server is responsible for serving the web-based application to clients. It handles incoming HTTP requests from the clients, processes them, and sends back appropriate responses. The server-side code of the application, such as GameService and related functionalities, will run on the web server.

Application Logic:

The application logic includes the core functionalities of the web-based game application, such as game management, team management, player interactions, and puzzle rendering. This logic is implemented on the server-side and is responsible for processing client requests and managing game sessions.

Data Storage:

The application may require a database to store game-related data, such as game instances, teams, and players. The data storage component is crucial for persisting game progress and user information. Common databases like MySQL, PostgreSQL, or NoSQL databases can be utilized.

Network Communication:

Network communication is essential for transmitting data between the clients and the web server. HTTP(S) is typically used for communication between the web browser and the server, allowing for stateless interactions.

## [Domain Model](#_8h2ehzxfam4o)

ProgramDriver Class:

This class serves as the entry point of the program, as indicated by the "main()" method.

It has a <<uses>> association with the "SingletonTester" class, suggesting that the ProgramDriver class utilizes the SingletonTester class in some capacity.

SingletonTester Class:

The SingletonTester class contains a method "testSingleton()" which is likely used for testing purposes.

There are no further attributes or associations provided in the diagram for this class.

Entity Class:

The Entity class represents a base class with common attributes that other classes can inherit from.

It has three attributes: "id" of type long, "name" of type String, and "Entity()" constructor to initialize the attributes.

The class provides getter methods "getId(): long" and "getName(): String" to access the attributes.

The "toString(): String" method is likely used to obtain a string representation of the object.

GameService Class:

The GameService class is a service class responsible for managing games.

It has several attributes: "games" (a list of Game objects), "nextGameId" of type long, "nextPlayerId" of type long, "nextTeamId" of type long, and "service" of type GameService (possibly used for implementing the Singleton pattern).

The "GameService()" constructor is provided to create an instance of the class.

The "getInstance(): GameService" method is a static method used to obtain the instance of the GameService class (Singleton pattern implementation).

The class includes methods to add and retrieve games, teams, and players, as well as get counts of certain entities.

Associations: The GameService class is connected to the Game class with "0...\*" cardinality, indicating that a GameService can manage multiple games.

Game Class:

The Game class holds information about a game and has an attribute "teams," which is a list of Team objects associated with the game.

The "Game(id: long, name: String)" constructor is provided to initialize a game with an id and a name.

The class includes a method "addTeam(name: String): Team" to add a team to the game and a "toString(): String" method for obtaining a string representation of the game.

Associations: The Game class is connected to the Team class.

Team Class:

The Team class represents a team in the game and has an attribute "players," which is a list of Player objects associated with the team.

The "Team(id: long, name: String)" constructor initializes a team with an id and a name.

The class includes a method "addPlayer(name: String): Player" to add a player to the team and a "toString(): String" method for obtaining a string representation of the team.

Associations: The Team class is connected to the Player class.

Player Class:

The Player class holds information about a player and has a "Player(id: long, name: String)" constructor to initialize a player with an id and a name.

The class includes a "toString(): String" method for obtaining a string representation of the player.

Object-Oriented Programming (OOP) Principles Demonstrated:

Inheritance:

The Entity class serves as a base class with common attributes like "id" and "name." The Game, Team, and Player classes are connected to the Entity class, indicating that they inherit these common attributes. Inheritance allows for code reuse and helps organize classes based on their shared characteristics.

Encapsulation:

The classes in the diagram use encapsulation to hide the implementation details of their attributes. Access to the attributes is provided through getter methods, ensuring controlled access and data integrity.

Association:

The diagram shows associations between classes using solid lines. For example, GameService is associated with Game, Game is associated with Team, and Team is associated with Player. Associations represent relationships between classes, allowing them to interact and collaborate.

Singleton Pattern:

The GameService class demonstrates the Singleton pattern with the static method "getInstance(): GameService." This pattern ensures that only one instance of the GameService class exists throughout the program's execution, fulfilling the requirement that only one instance of the game exists in memory at any given time.

**"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 known for their stability and reliability. They provide a Unix-based environment that is well-suited for web-based software applications. Mac servers generally offer good performance and are preferred for hosting web applications due to their strong security features and ease of use. However, one potential drawback is the higher cost of Mac hardware compared to other server options.  Characteristics:  macOS can also be used to host web-based applications using web server software like Apache or Nginx.  It provides a Unix-based environment, which is similar to Linux, offering stability and reliable performance.  Advantages:  macOS is known for its user-friendly interface and ease of use, which can be beneficial for server management.  For developers already working on macOS, hosting the application on macOS might simplify the development and deployment process.  Weaknesses:  The licensing costs for macOS can be higher compared to Linux, especially when deploying on multiple servers.  Hardware costs for Mac servers might be higher compared to commodity hardware options available for Linux.  Potential Licensing Costs for Server Operating System:  macOS is a commercial operating system developed by Apple Inc. As such, it is not available for free.  To use macOS as a server operating system, the client would need to purchase licenses for macOS Server. macOS Server is a separate software package from the standard macOS and includes additional server-related features.  The cost of macOS Server licenses might vary based on the version and the number of server instances required by the client. As of my last update in September 2021, the pricing for macOS Server was available through Apple's website or authorized resellers | Linux servers are widely used for web hosting due to their open-source nature and robust performance. They offer excellent stability, security, and scalability, making them suitable for hosting web-based software applications. Linux servers are also cost-effective as there are no licensing fees, and they can run on a wide range of hardware configurations. However, managing Linux servers may require more technical expertise compared to other platforms.  Characteristics:  Linux is a popular choice for hosting web-based applications due to its stability, security, and scalability.  It offers a wide range of web server options, such as Apache, Nginx, and Lighted, which are well-suited for serving web content efficiently.  Linux provides robust networking capabilities, making it suitable for handling multiple concurrent connections from thousands of players.  Advantages:  Linux is an open-source operating system, which means there are no licensing costs for the OS itself, making it a cost-effective choice.  Its modular architecture allows for easy customization and optimization, enabling efficient  utilization of server resources.  Linux is known for its strong security features, reducing the risk of security breaches.  Weaknesses:  Managing Linux servers may require more technical expertise compared to other platforms, potentially leading to higher administration costs.  Compatibility with certain proprietary software or applications developed specifically for Windows may be a challenge.  Potential Licensing Costs for Server Operating System:  Linux is an open-source operating system, which means it is generally available for free. Most Linux distributions, such as Ubuntu, CentOS, Fedora, and Debian, do not require any licensing fees.  However, some enterprise-grade Linux distributions may offer additional support and maintenance services, and these might come with associated costs. The extent of support and licensing fees will depend on the specific Linux distribution and the level of support required by the client. | Windows servers are commonly used for hosting web applications, especially those developed using Microsoft technologies. They provide a familiar environment for developers who are already experienced with Windows-based systems. However, Windows servers may have higher licensing costs, and they are generally considered to be more vulnerable to security threats compared to Mac and Linux servers.  Characteristics:  Windows servers are commonly used for hosting web applications, especially those developed using Microsoft technologies.  Microsoft provides Internet Information Services (IIS) as the built-in web server for hosting web applications on Windows servers.  Advantages:  Windows offers seamless integration with Microsoft technologies and tools, making it suitable for applications developed using .NET and C#.  The availability of familiar development tools like Visual Studio can streamline the development process.  Weaknesses:  Windows servers might have higher licensing costs compared to Linux or macOS, especially for large-scale deployments.  Windows servers are often perceived to be more vulnerable to security threats compared to Linux.  Potential Windows Server is a commercial operating system developed by Microsoft, specifically designed for server deployments.  The licensing costs for Windows Server can vary based on the edition (Standard, Datacenter, Essentials, etc.) and the number of server cores or users/devices being used.  Microsoft typically offers Windows Server licenses through volume licensing programs, such as the Microsoft Open License Program or the Microsoft Enterprise Agreement, which allow organizations to purchase licenses in bulk at discounted rates. | Mobile devices are not typically used as server platforms for web-based software applications. Instead, they act as clients that interact with server-side applications. Hosting a web-based software application on mobile devices would be impractical due to their limited processing power, memory, and connectivity capabilities. |
| **Client Side** | Developing client-side software for Mac involves considering the macOS operating system's specific user interface guidelines and design principles. It requires expertise in macOS app development, which can be achieved using programming languages like Swift or Objective-C. Development tools like Xcode, Apple's integrated development environment (IDE), are commonly used for Mac application development. The cost and time involved in developing for Mac may be influenced by the complexity of the application and the availability of skilled Mac developers. | Developing client-side software for Linux involves catering to various Linux distributions and desktop environments, each with its own user interface standards. Developers must have expertise in Linux app development using languages like C++, Python, or GTK. IDEs like Qt Creator or Eclipse are commonly used for Linux application development. The cost and time considerations for Linux development may depend on the target distributions and the need for cross-platform compatibility. | Developing client-side software for Windows involves adhering to the Windows operating system's user interface guidelines and design conventions. Common programming languages for Windows app development are C# and .NET. Microsoft's Visual Studio is a widely used IDE for Windows application development. The cost and time required for Windows development may vary depending on the complexity of the application and the familiarity of the developers with Windows technologies. | Developing client-side software for mobile devices requires expertise in mobile app development for specific platforms like iOS or Android. For iOS, developers use Swift or Objective-C, while Java or Kotlin is commonly used for Android development. IDEs like Xcode (for iOS) and Android Studio (for Android) are used for mobile app development. The cost and time involved in mobile app development may be influenced by the platform's market share, target audience, and complexity of the application. |
| **Client Slide- Multiple operating system.** | Multiple Operation System approach, Android Mobile , IOS, Desktop  Cost & Time  hypothetically speaking, let's consider a scenario where the client wants to support three client platforms: web browsers (Linux, Mac, Windows), iOS, and Android. We'll assume that the project scope includes creating a modern, responsive HTML interface for web browsers and native mobile apps for iOS and Android.  Cost Estimate (Hypothetical):  Development Team: Assuming a team of 6 developers with varying expertise in web and mobile development, the average monthly salary for each developer is $6,000 USD. Let's consider a development period of 6 months.  Total cost for development team: 6 developers \* $6,000 \* 6 months = $216,000 USD  Development Tools: The development team may require licenses for development tools and frameworks. Assuming an average cost of $1,000 USD per developer for tools and licenses.  Total cost for development tools: 6 developers \* $1,000 = $6,000 USD  Server Hosting: To host the web-based application, the client may opt for cloud hosting services. Assuming a cost of $500 USD per month for cloud hosting services for 6 months.  Total cost for server hosting: $500 \* 6 months = $3,000 USD  Miscellaneous Expenses: Additional expenses for testing, design assets, and other project-related costs. Assuming $10,000 USD for miscellaneous expenses.  Total Estimated Cost: $216,000 + $6,000 + $3,000 + $10,000 = $235,000 USD  Time Estimate (Hypothetical):  Web Application Development: Assuming it takes 3 months to develop the web-based application for desktop clients.  iOS App Development: Assuming it takes 2 months to develop the iOS app.  Android App Development: Assuming it takes 2 months to develop the Android app.  Testing and Optimization: Allocating 1 month for testing and optimizing the application for all platforms.  Total Estimated Time: 3 months (Web) + 2 months (iOS) + 2 months (Android) + 1 month (Testing) = 8 months | Supporting multiple types of clients, including web browsers on desktop platforms (Linux, Mac, Windows) and mobile devices (iOS and Android), requires careful consideration in the application development process. To ensure the application is compatible with all these platforms, the following software development considerations need to be taken into account:  Cross-Platform Compatibility:  To achieve cross-platform compatibility, the application should be developed using web technologies that are universally supported by modern web browsers on different platforms. This includes using HTML, CSS, and JavaScript to create the user interface and functionality.  Adopting responsive web design principles is crucial to ensure that the application adapts to various screen sizes and resolutions, providing an optimal user experience across desktop and mobile devices.  Mobile-First Approach:  Given that the application aims to support mobile devices, a mobile-first approach should be adopted during the development process. This involves designing the user interface and user experience primarily for mobile devices and then progressively enhancing it for larger screens.  Testing and Debugging on Multiple Platforms:  Thorough testing and debugging should be conducted on various web browsers and operating systems to identify and resolve compatibility issues. This includes testing on popular web browsers like Chrome, Firefox, Safari, and Edge, as well as different versions of each browser.  Additionally, testing on multiple mobile devices with varying screen sizes and operating systems (iOS and Android) is essential to ensure consistent performance.  Leveraging Web Standards and Frameworks:  Using web standards and frameworks can streamline the development process and ensure better compatibility across different platforms. For example, utilizing CSS frameworks like Bootstrap or Foundation can help create responsive layouts, and using JavaScript frameworks like React or Vue.js can simplify application development.  API Integration and Back-End Compatibility:  The application needs to communicate with the back-end web application running on the server. This requires designing a robust and well-documented RESTful API that is compatible with various client platforms.  The API should be designed to handle requests from different devices, including desktop browsers and mobile applications, and respond with appropriate data in a consistent format.  Performance Optimization:  Optimizing the application's performance is crucial for delivering a smooth and responsive experience on both desktop and mobile devices. Techniques such as code minification, image compression, and caching should be employed to reduce load times.  Team Expertise and Skillsets:  The development team should have expertise in web development, including proficiency in HTML, CSS, JavaScript, and responsive design principles.  For mobile platforms, developers with experience in mobile app development (iOS and Android) using technologies like Swift, Objective-C, Java, or Kotlin will be necessary.  Development Timeline and Costs:  Supporting multiple client platforms may require additional development time and resources. The project timeline and budget should be carefully planned to accommodate the complexity of multi-platform development. | | |
| **Development Tools** | For developing software on Mac, popular programming languages include Swift (for macOS and iOS development) and Objective-C (mostly for macOS). Xcode is the primary IDE used for macOS and iOS application development on Mac. | For software development on Linux, developers commonly use languages like C, C++, Python, and Java, among others. IDEs like Eclipse, Visual Studio Code, and IntelliJ IDEA are popular choices for Linux development. | Windows development typically involves using languages like C#, VB.NET, and C++. Microsoft's Visual Studio is the primary IDE used for Windows application development | Mobile app development for iOS (Swift or Objective-C) and Android (Java or Kotlin) is typically done using Xcode (for iOS) and Android Studio (for Android) as the primary IDEs. |
| **Development tool / Multiple operating system.** | Multiple Operation System approach, Android Mobile , IOS, Desktop  Impact on Development Team:  Supporting multiple client platforms may require developers with expertise in different programming languages and frameworks. A single development team with diverse skill sets can handle web and mobile development for smaller projects. However, for larger and more complex projects, multiple specialized teams may be needed to ensure efficiency and expertise in each platform.  Licensing Costs:  Some development tools, IDEs, and libraries may have licensing costs. For example, the licenses for IntelliJ IDEA and WebStorm (IDEs) have a subscription-based model with different pricing tiers. However, Visual Studio Code is open-source and free to use. Android Studio and Xcode are free IDEs.  Additional licensing costs may arise if the team uses premium plugins or libraries that have commercial licenses. It's essential to consider the licensing costs while selecting development tools and libraries to stay within the project's budget. Open-source alternatives can also be used to reduce licensing expenses. | Web-Based Application (Linux, Mac, Windows):  Programming Languages: HTML, CSS, JavaScript are the core languages for developing the frontend of the web-based application. Additionally, backend development can be done using languages like Java, Python, or Node.js.  Integrated Development Environments (IDEs): Developers can use various IDEs such as Visual Studio Code, IntelliJ IDEA, Eclipse, or WebStorm for coding, debugging, and managing the project.  Version Control: Git is commonly used for version control to manage code changes and collaborate efficiently.  Frameworks and Libraries: Frameworks like React, Angular, or Vue.js can be utilized for building the frontend, while backend frameworks like Spring Boot or Django can be used for server-side development.  Database: For data storage, developers can use databases like MySQL, PostgreSQL, or MongoDB.  iOS Application:  Programming Language: Swift is the primary programming language for iOS app development.  Integrated Development Environment (IDE): Xcode is the official IDE provided by Apple for iOS app development.  Version Control: Git can be used for version control in iOS development as well.  Frameworks and Libraries: Apple's UIKit framework is commonly used for building iOS user interfaces and interactions.  Deployment: For deploying iOS apps to the App Store, developers need to be enrolled in the Apple Developer Program.  Android Application:  Programming Language: Java or Kotlin can be used for Android app development. Kotlin is the preferred language for new projects.  Integrated Development Environment (IDE): Android Studio is the official IDE provided by Google for Android app development.  Version Control: Git can be used for version control in Android development too.  Frameworks and Libraries: Android developers can use Android SDK and libraries like Retrofit for networking, Room for local database storage, etc. | | |

## Recommendations

By leveraging the capabilities of Amazon Web Services (AWS) as the recommended operating platform, this strategic approach will enable "Draw It or Lose It" to seamlessly extend its reach across platforms, while safeguarding user data, maintaining performance, and fostering a dynamic gaming experience. This document further delves into the intricate details of each component, offering insights and recommendations to empower The Gaming Room's expansion aspirations.

1. Operating Platform: For expanding "Draw It or Lose It" to various computing environments, I recommend utilizing a cloud-based operating platform. Cloud platforms offer scalability, flexibility, and the ability to deploy the game across different operating systems while efficiently managing resources. One strong option is Amazon Web Services (AWS), specifically using Amazon Elastic Compute Cloud (EC2) for hosting the game's server-side components.
2. Operating Systems Architectures: AWS operating platform architecture will leverage a combination of EC2 instances, managed services, microservices, security measures, and monitoring tools to provide a scalable, reliable, and secure foundation for expanding "Draw It or Lose It" across various operating systems and platforms.

Amazon Elastic Compute Cloud (EC2) Instances:

EC2 instances are virtual machines that provide computer capacity in the cloud. In the context of "Draw It or Lose It," EC2 instances would host the server-side components of the game. Different types of EC2 instances can be selected based on the game's resource requirements, such as CPU, memory, and GPU capabilities.

Elastic Load Balancing:

Elastic Load Balancing (ELB) distributes incoming traffic across multiple EC2 instances to ensure even distribution and prevent overloading of any single instance. This enhances the game's availability and responsiveness by efficiently managing user requests.

Amazon Relational Database Service (RDS):

RDS offers managed database services that support different database engines like MySQL, PostgreSQL, or Microsoft SQL Server. The game's user profiles, saved game data, and other relevant information can be stored in an RDS instance, ensuring data integrity and durability.

Amazon S3 for Storage:

Amazon S3 is used to store static assets, game resources, and other large files. These files can be accessed securely by the game's client applications running on various platforms. S3's global content distribution capabilities ensure low latency for users around the world.

Microservices Architecture:

The architecture can be designed using microservices, where different game functionalities are broken down into smaller, modular services. Each microservice performs a specific function, such as user authentication, gameplay logic, and social features. These services can communicate via APIs and can be developed, deployed, and scaled independently.

API Gateway:

API Gateway acts as a frontend to the microservices, allowing clients to interact with different components of the game via well-defined APIs. It handles request routing, authentication, and authorization, ensuring that only authorized clients can access the game's services.

Amazon CloudFront:

Amazon CloudFront is a content delivery network (CDN) that accelerates the distribution of dynamic and static content by caching it at edge locations. This enhances the game's performance and reduces latency for users by serving content from the nearest edge location.

Security Measures:

AWS's security measures, such as IAM, encryption options, security groups, and network ACLs, are implemented to ensure the confidentiality, integrity, and availability of the game's data and services. Access to resources is tightly controlled, and encryption is used to protect sensitive data at rest and in transit.

Monitoring and Analytics:

Amazon CloudWatch is used for monitoring the health and performance of various AWS resources, including EC2 instances, RDS databases, and API Gateway. CloudWatch provides metrics, logs, and alarms that help in identifying issues and optimizing resource utilization.

Auto Scaling:

Auto Scaling dynamically adjusts the number of EC2 instances based on traffic load. This ensures that the game can handle spikes in user activity without manual intervention while optimizing costs during periods of lower demand.

Deployment and Management:

AWS provides tools like AWS Management Console, AWS Command Line Interface (CLI), and Infrastructure as Code (IaC) services like AWS CloudFormation to manage the deployment, configuration, and updates of the entire architecture.

1. Storage Management: For storage management, Amazon S3 (Simple Storage Service) can be employed to store and manage game-related data such as user profiles, saved games, and assets. This distributed storage system offers high durability, availability, and ease of integration with cloud-based services.

Scalability and Durability:

Amazon S3 is designed for high durability and availability. It stores data redundantly across multiple data centers and automatically handles replication and data distribution.

Object-Based Storage:

Amazon S3 uses an object-based storage model, allowing you to store and retrieve files as objects. Each object includes data, metadata, and a unique key. This structure is well-suited for storing various types of game-related files, such as images, videos, audio files, and user-generated content.

Global Content Distribution:

Amazon S3 integrates seamlessly with Amazon CloudFront, a content delivery network (CDN). By using CloudFront, you can distribute game assets and content to edge locations around the world.

Pay-as-You-Go Model:

Amazon S3 follows a pay-as-you-go pricing model, meaning you only pay for the storage you use and the data transferred. This aligns well with the scalability requirements of "Draw It or Lose It."

1. Memory Management: The recommended operating platform, such as Amazon EC2, employs memory management techniques like virtual memory and dynamic memory allocation. These techniques allow the game to efficiently allocate and manage memory resources, ensuring optimal performance and preventing memory-related issues.
2. Distributed Systems and Networks:

To achieve communication between various platforms, a microservices architecture can be adopted. Each game component, like gameplay logic, user profiles, and social features, can be developed as independent microservices. These microservices can communicate via well-defined APIs and protocols, enabling seamless interaction between platforms. For network connectivity, the game can utilize HTTPS and WebSocket protocols for secure and real-time communication.

In cases of connectivity issues or outages, implementing redundancy and failover mechanisms will be crucial. Load balancers and distributed databases can be employed to ensure high availability and reliability of the game's services.

1. Security:

Ensuring security for user information is paramount. The recommended operating platform, AWS, offers robust security capabilities. Identity and Access Management (IAM) can be used to control access to resources and enforce least privilege principles. Data in transit can be encrypted using protocols like TLS/SSL, and sensitive data at rest can be protected using server-side encryption in Amazon S3.

Here's how AWS can assist with security:

Identity and Access Management (IAM):

AWS IAM allows you to manage user identities and their permissions to access AWS resources. You can create and manage user accounts, assign permissions using policies, and enforce the principle of least privilege. This ensures that users only have access to the resources they need for their roles, reducing the attack surface.

Encryption:

AWS offers various encryption options to protect data both in transit and at rest. Transport Layer Security (TLS) can be used to encrypt data in transit, while services like Amazon S3 and Amazon RDS support server-side encryption to safeguard data at rest.

Network Security:

AWS provides Virtual Private Cloud (VPC), which allows you to isolate your resources within a private network, control inbound and outbound traffic with security groups and network ACLs (Access Control Lists), and set up VPNs or Direct Connect for secure connectivity.

DDoS Protection:

AWS Shield provides Distributed Denial of Service (DDoS) protection to safeguard your applications against large-scale, sophisticated attacks, helping to maintain availability and uptime.

Security Groups and NACLs:

AWS allows you to define security groups for EC2 instances and network ACLs for VPCs. These act as virtual firewalls, controlling inbound and outbound traffic and helping to prevent unauthorized access.

Web Application Firewall (WAF):

AWS WAF helps protect web applications from common web exploits and vulnerabilities. It allows you to define rules to filter and monitor incoming web traffic, blocking potential threats.

Security Best Practices:

AWS provides best practice guidelines, whitepapers, and documentation on how to architect secure applications, from design to deployment.

1. Conclusion:

By leveraging a cloud-based operating platform like Amazon EC2, adopting a microservices architecture, and incorporating strong security measures, "Draw It or Lose It" can be effectively expanded to various operating systems and platforms while ensuring smooth communication and protecting user information. This approach provides the scalability, flexibility, and security needed to support the game's growth and user engagement across different computing environments.