

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 | 01/28/24 | Devin Burdick | Created SDT |
| 1.1 | 02/10/24 | Devin Burdick | Continued development of SDT in Evaluation section |
| 1.2 | 02/24/24 | Devin Burdick | Continued development of SDT in Recommendation section |

**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)

Our software design problem is to develop a web-based game application named Draw It or Lose It for The Gaming Room. The client requested that the software has the ability to support one or more teams, multiple players in a team, different unique team and game names, and only one instance of each game in memory at any time. The web based app is also based on their Android app.

## Requirements

Provided by the client:

* 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.

## [Design Constraints](#_2et92p0)

The design constraints for developing the game application in this environment include the following:

1. The application must be compatible with the many different web browsers available today.
2. The application must be able to support the creation and management of multiple teams that consist of multiple players.
3. The application must be able to enforce unique game and team names.
4. The application must generate and use unique identifiers for each instance of games, teams, and players.
5. The application must allow only one instance of the game to exist in memory at any given time.

The many design constraints listed above foster an enormous amount of implications for development including the need for efficient data structures, adopting security best practices and following coding best standards and practices, testing on various different web browsers, and selecting frameworks, technologies, and algorithms correctly.

## [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 UML class diagram provided below represents the domain model within the software design. The many different classes include Entity, Game, Player, and Team. The entity class is the parent class for Game, Player, and Team which indicates that they share common attributes and behaviors and inherit from one another.

The 0..\* relationship between GameService and Game, Game and Team, and Team and Player are as follows below:

1. GameService has a 0..\* relationship with Game, meaning that a GameService object can be associated with multiple Game Objects.
2. Game has a 0..\* relationship with Team, meaning that a Game object can be associated with multiple Team Objects.
3. Team has a 0..\* relationship with Player, meaning that a Team object can be associated with multiple Player objects.

In summary, the UML class diagram demonstrates inheritance relationships between Entity, Game, Player, and Team, and highlights the 0..\* relationships between GameService and Game, Game and Team, and Team and Player. The diagram also uses encapsulation with getters and setters. This allows to keep the fields private while providing access through public methods. The getters and setters in the UML diagram include:

**GameService Class:**

getInstance(): GameService

addGame(name:String): Game

getGame(id:long): Game

getGame(name:String): Game

getGameCount(): int

getNextPlayerId(): long

getNextTeamId(): long

**Game Class:**

Game(id: long, name: String)

addTeam(name:String): Team

toString(): String

**Team Class:**

Team(id: long, name: String)

addPlayer(name:String): Player

toString(): String

**Player Class:**

Player(id: long, name: String)

toString(): String

**"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 offers a reliable platform that supports server-based deployment for hosting web applications. Its advantages include stability being improved by the Unix-based infrastructure, and website hosting is made easier by the availability of tools like Apache and Nginx. However, it’s weaknesses include that the Gaming Room may have to take the cost of the macOS server license into account, and it has limited scalability for large-scale deployments. | Because of its wide command-line capabilities and open-source nature, Linux’s advantages make it a preferred option for server-side operations. It is very configurable and performs exceptionally well while running server applications. Because Linux has robust security mechanisms, it can be used to host important services. Linux’s weaknesses include that it might, however, lack intuitive graphical user interface (GUI) capabilities for server management, which in return requires a solid command-line operations knowledge base and a steeper learning curve. Linux’s open-source nature makes it very cost-effective as well. | Server-side programs run more easily on Windows, which opens up the platform to a wider user base. It has a good range of server management tools with user-friendly graphical user interfaces and supports Java applications. However, Window’s weaknesses include that when compared to Unix-based systems, Windows can have reliability issues, and its licensing fees might be an issue and come at a higher cost. | Java programs are not commonly deployed on mobile devices as server platforms. The Draw It or Lose It! application's server-side components cannot be hosted on them due to resource constraints, scalability challenges, and security concerns.  Mobile devices advantages include that they are more accessible to a wide user base. However, its weaknesses include that it has limited resources compared to dedicated servers and backend infrastructure still needs to be hosted on a traditional server as well. |
| **Client Side** | JavaScript and HTML are used to create web-based clients on Mac computers. The default browser, Safari, offers good compatibility and a seamless programming environment. Cross-browser compatibility testing, however, could be required for other browsers. The cost of the Mac development tools can be relatively expensive and their may be licensing fees for certain technologies. Time includes learning the Mac development environment and testing on different versions of the MacOS. Finally the necessary expertise includes proficiency in tools like Xcode and understanding Mac-specific design guidelines. | Because Linux supports a wide range of web browsers and development tools, it is an excellent choice for client-side programming. Customization is made easier by the open-source nature, and developers can test their applications on a variety of browsers. Linux, however, may encounter difficulties with some proprietary software and drivers for particular hardware parts. The cost of Linux is lower in terms of licensing but may have potential costs for specific tools or libraries. Time includes doing configuring and testing on many different Linux distributions. Finally, required expertise includes proficiency in Linux development and knowledge of many diverse Linux environments. | Windows has a large market share and is frequently used for client-side development. It offers comprehensive support for developer tools and web browsers. JavaScript and HTML work together flawlessly in Windows systems. However, client apps may have security issues because of the system's vulnerability to viruses and malware. The cost of Windows includes licensing fees for development tools and costs with Window’s technologies. The time includes extensive testing across different versions of Windows and integration with Window’s specific features. Finally, the expertise necessary is proficiency in Visual Studio and understanding Window’s specific design guidelines. | The main focus of the Draw It or Lose It! client application is mobile devices. When creating a web client for mobile devices, factors like screen size variations, touch interactivity, and responsiveness must be taken into account. To guarantee a consistent user experience across various mobile platforms and browsers, compatibility testing is essential. The cost’s would include licensing fees for the different mobile development platforms and gaining access to various devices for testing. The time is very extensive testing across the different operating systems and adapting different screen sizes, etc. Finally, the expertise includes proficiency in mobile development languages and understanding mobile-specific design guidelines as well. |
| **Development Tools** | Mac provides a full suite of Java, HTML, and JavaScript development tools. Package managers make dependency management easier, and the Xcode integrated development environment is well respected for Java development. It might, however, be devoid of some customization and flexibility features found in other systems. Specifically, Mac uses the programming language Swift primarily, paired with IDE Xcode which requires an expertise of the development team in Swift. Xcode is also free but could include costs down the line when distributing the application. Coordination and cooperation is necessary within the team. | A powerful development environment is offered by Linux, and package managers make a large selection of tools available. It is excellent at assisting with open-source development, and the command-line interface is quite useful for activities related to development. Some commercial IDEs and development tools, however, might only offer a restricted amount of native Linux compatibility. Specifically, Linux primarily includes the languages C, C++, Python, and Java, paired with Visual Studio Code, Eclipse, or GNU Compiler Collection as the IDE. The expertise required for Linux includes diverse skills across many different languages and cooperation between potentially different teams focusing on different languages. | With well-known integrated development environments (IDEs) like Visual Studio and broad support for Java, HTML, and JavaScript, Windows provides an intuitive development environment. The platform offers a large library selection and seamlessly connects with a variety of version control systems. But some development tools might have license fees, and background operations could affect how well the system performs. Specifically, Window’s includes the progamming languages C#, C++, and .NET languages paired with the IDE Visual Studio or .NET Core. The expertise necessary includes Proficiency in C# and .NET as well as coordination between frontend and backend teams. | Platform-specific tools are used in mobile development. Xcode is the main IDE for developing iOS apps on Mac computers. Android Studio can be used to develop Android applications on Linux or Windows. While cross-platform tools such as React Native and Xamarin offer possibilities for multi-platform development, they might not be as flexible as native development environments. There may be licensing fees from Apple or Google Play and depending on the cross-platform framework there may be licensing fees as well. Specifically, mobile devices includes Swift, Kotlin and JavaScript paired with Xcode, Android Studio, and Visual Studio. The expertise required includes knowledge in each of the above and coordination between iOS, Android, and potentially even more development teams. |

## 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:** For the expansion of "Draw It or Lose It" to multiple platforms, I recommend adopting a cloud-based operating platform that can seamlessly support both server-side hosting and client-side deployment. Considering the evaluation of traditional operating platforms (Linux, Mac, Windows) and mobile platforms (iOS, Android), the use of a cloud service like Amazon Web Services (AWS) is highly recommended. AWS provides the scalability, flexibility, and compatibility required for hosting and deploying the game across various environments.
2. **Operating Systems Architectures:** Given the diverse set of platforms, the chosen operating platform (AWS) employs a cloud-based architecture. This involves server instances that can run different operating systems, ensuring compatibility with Linux, Mac, Windows, iOS, and Android. The cloud-based architecture provides the necessary scalability to handle thousands of players concurrently.
3. **Storage Management:** For efficient storage management, I recommend utilizing Amazon Simple Storage Service (S3) for hosting static assets and Amazon Relational Database Service (RDS) for structured data storage. S3 ensures fast and reliable access to images and other resources, while RDS offers a scalable and managed database solution, ensuring data integrity and optimal performance.
4. **Memory Management:** AWS provides various memory management features that cater to the requirements of "Draw It or Lose It." Elastic Load Balancing (ELB) ensures even distribution of traffic, optimizing memory utilization. Additionally, services like Amazon ECS and AWS Lambda efficiently manage memory resources for containerized applications, contributing to the overall performance of the game.
5. **Distributed Systems and Networks:** To facilitate communication between various platforms, a microservices architecture deployed on AWS can be employed. Services like Amazon API Gateway and AWS Lambda enable seamless interactions between different components. Amazon Virtual Private Cloud (VPC) ensures secure networking between components, and the use of AWS Auto Scaling enhances connectivity robustness by dynamically adjusting resources based on demand.

In a distributed system, we always must consider the possibility of outages. I recommend implementing automatic failover, health monitoring systems, and load balancing. Load balancing will allow us to distribute the workload evenly across many instances, automatic failover will redirect traffic to a healthy instance in the case of an outage, and health monitoring will check the status of our system and alert administrators of any issues.

1. **Security:** Security may be the most critical aspect of our web-based application. I recommend implementing a security approach that consists of multiple layers. First, encryption is necessary, which can be accomplished through AWS Key Management Services for data that is at rest and in transit. Next, implementing identity and access management or IAM with access controls to ensure authorized access to resources is necessary. The use of virtual private cloud security groups and network access control lists to control inbound and outbound traffic is also necessary. Finally, utilizing a monitoring and logging system such as AWS CloudWatch in our system for real-time monitoring and AWS CloudTrail for logging and auditing which ensures early detection of security threats will keep our web application safe.

In conclusion, I recommend that The Gaming Room expand their Draw it or Lose it game onto different

platforms using the cloud-based sever platform AWS. The cloud-based architecture can run many

instances to support the required different operating platforms. For the storage necessities, I

recommend the Amazon RDS, which is a relational database management system, paired with the

virtual memory management such as Amazon ECS and AWS Lambda. To address the necessary

distributed systems and networking, I recommend microservices architecture with an API gateway

which can be accomplished with services like Amazon API and then Amazon Virtual Private Cloud for

secure networking. Finally, built-in security features are necessary with encryption, identity and access

management, and virtual private cloud security groups and network access control lists. This level of

security can be accomplished with AWS Key Management Services for the encryption as well as AWS

CloudWatch and AWS CloudTrail to monitor the system in real time and log and audit which allows for

early detection of security vulnerabilities as well. All of the above recommendations meet The Gaming

Room’s needs to ensure a positive user side experience and sufficient client system architecture.