

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 | 01/22/2023 | Jarrod Thomisee | Initial Proposal |
| 2.0 | 02/05/2023 | Jarrod Thomisee | Updates to evaluation and recommendations |
| 3.0 | 02/07/2023 | Jarrod Thomisee | Updates to recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The software design problem at hand is the development of a web-based version of the game "Draw It or Lose It". The game must meet specific requirements, such as the ability to have one or more teams involved, each team having multiple players assigned to it, and that game and team names must be unique. In addition, only one instance of the game can exist in memory at any given time.

The solution proposed is to create a database to store information about games, teams, and players. A user interface for creating and joining games will be developed, including the ability for users to choose unique game and team names, and to assign players to teams. A game engine will be developed to handle multiple teams and players, track the progress of each round, and handle the logic for allowing remaining teams to offer guesses after time expires. A rendering system will be developed to display the stock drawings as clues to the players, and a system for generating unique identifiers for each instance of a game, team, or player will be implemented to ensure that only one instance of the game exists in memory at any given time. The application will be thoroughly tested and documented in a software design document for review by the technical manager.

It's important to note that the hardware requirements will depend on the decisions made during the software development process.

## [Design Constraints](#_2et92p0)

Design constraints for developing the game application in a web-based distributed environment include:

Scalability: The game application must be able to handle a large number of concurrent users and games and be able to handle an increase in traffic without significant performance degradation. This may require the use of load balancers, caching, and other performance optimization techniques.

Security: The game application must protect sensitive data, such as user information, from unauthorized access and ensure that all communication between the client and the server is secure. This may require the use of encryption, authentication, and authorization mechanisms.

Reliability: The game application must be able to handle network failures and other types of errors and be able to recover gracefully. This may require the use of load balancers, replication, and failover mechanisms.

Latency: The game application must be able to handle the delay caused by the distance between the client and the server and ensure that the game play experience is not negatively impacted. This may require the use of content delivery networks (CDNs), and other latency optimization techniques.

Compatibility: The game application must be compatible with different web browsers and devices and must be able to adapt to different screen sizes and resolutions. This may require the use of responsive design techniques, and cross-browser and cross-device testing.

Meeting these constraints will require the use of multiple technologies and techniques, such as load balancers, caching, encryption, replication, failover, content delivery networks (CDNs), responsive design, and cross-browser and cross-device testing.

## [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 classes in the UML class diagram for the Draw It or Lose It game project are related to each other as follows:

The Entity class stores the name and Id of all game entities including, Games, Players, and Team.

GameService class has a Game class: The GameService class keeps track of all the instances of the Game class.

Game class extends Entity and has a Team class: The Game class has a collection of Team objects, representing the teams participating in the game. The Game class also has methods for creating and managing the game, such as adding Teams.

Team class extends Entity and has a Player class: The Team class has a collection of Player objects, representing the players assigned to the team. The Team class also has methods for creating and managing teams, such as adding players to the team.

The Player class extends Entity and has methods for creating and getting information about a Player.

These classes demonstrate several object-oriented programming (OOP) principles, such as:

Encapsulation: Each class encapsulates its own data and behavior, hiding the implementation details from other classes. This makes the code more maintainable and easier to understand.

Inheritance: The classes are organized in a hierarchical structure, with the Game, Team, and Player classes inheriting from a common base class or interface. This allows for code reuse and a more organized structure.

Polymorphism: The classes have methods with the same name but with different implementations, allowing them to be used interchangeably and making the code more flexible.

Abstraction: The classes abstract the complex functionality of the game into smaller, more manageable units. This makes the code more modular and easier to understand.

By using these OOP principles, the design of the game application is made more efficient, maintainable, and scalable. The encapsulation allows for hiding the implementation details and make the code more maintainable and easier to understand, the inheritance allows for code reuse and a more organized structure, the polymorphism allows for the classes to be used interchangeably and making the code more flexible and the abstraction allows for breaking down the complexity of the game into smaller units making the code more modular and easier to understand.

**"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 is a proprietary OS developed and owned by Apple. It is based on a Unix-like architecture and is known for its stability and security. Advantages of Mac include its user-friendly interface, built-in development tools, and a large number of available libraries and frameworks. However, Mac can be more expensive than other OS and the hardware and software are only available from Apple. | Linux is an open-source OS that is based on a Unix-like architecture. It is known for its stability, security, and flexibility, and is widely used for web servers and cloud-based applications. Advantages of Linux include its low cost, a large community of developers, and a wide range of available libraries and frameworks. However, Linux can have a steeper learning curve than other OS and may not have as many user-friendly tools as Mac or Windows. | Windows is a proprietary OS developed and owned by Microsoft. It is widely used for desktop and laptop computers, and is known for its user-friendly interface and built-in development tools. Advantages of Windows include its wide availability and compatibility with many different software and hardware products. However, Windows can be more vulnerable to security threats than other OS, and some developers may find it less flexible and customizable than Linux or Mac. | Mobile devices are a convenient option for hosting a web-based application due to their portability and accessibility, but their small screens, limited processing power, and memory can make it difficult to run complex or resource-intensive web applications. Additionally, mobile devices can be less stable and secure than desktop or laptop computers, so careful consideration should be given when deciding to use them for hosting a web-based application. |
| **Client Side** | Supporting Mac clients requires careful consideration of cost, time, and expertise. The cost of development may increase with the need to support Mac-specific tools and libraries, and developers may need to purchase Mac hardware to test the application. The development time may also increase as the application needs to be tested and debugged on Mac hardware. Additionally, developers will need to have expertise in Mac-specific development platforms and languages. | The cost of development may be less expensive than other operating systems, but developers may need to purchase Linux hardware to test the application. The development time may also increase as the application needs to be tested and debugged on Linux hardware. Additionally, developers will need to have expertise in Linux-specific development platforms and languages. | Supporting Windows clients requires careful consideration of cost, time, and expertise. The cost of development may increase with the need to support Windows-specific tools and libraries, and developers may need to purchase Windows hardware to test the application. The development time may also increase as the application needs to be tested and debugged on Windows hardware. Additionally, developers will need to have expertise in Windows-specific development platforms and languages. | When developing a web-based software application that needs to support mobile devices, the software development considerations that are necessary include cost, time, and expertise. Supporting mobile devices can increase the development cost due to the need for mobile device-specific development tools and libraries, as well as the need to purchase multiple types of mobile devices for testing. Additionally, supporting mobile devices can also increase the development time, as more time is needed for testing and debugging on different types of mobile devices. To support mobile devices, developers need expertise in mobile device-specific development platforms and languages, and knowledge of the different development tools and libraries required for different types of mobile devices. |
| **Development Tools** | Web-based software application for deploying on Mac can be built using several relevant programming languages such as HTML, CSS, and JavaScript for the front-end and Ruby, Python, and PHP for the back-end. IDEs such as Xcode and Visual Studio Code can be used for development and Git for version control. Node.js, Homebrew, and other tools can be used to manage libraries and dependencies. | Web-based software application for deploying on Linux can be built using several relevant programming languages such as HTML, CSS, and JavaScript for the front-end and Python, Ruby, and PHP for the back-end. IDEs such as Eclipse, PyCharm can be used for development, Linux package managers and Git for managing dependencies and version control. Node.js can also be used to run JavaScript code on the server-side. | Web-based software application for deploying on Windows can be built using several relevant programming languages, IDEs such as Visual Studio or Eclipse, package managers, and other tools such as Git and Node.js. | Web-based software application for deploying on Mobile Devices can be built using several relevant programming languages, IDEs such as Android Studio or Xcode, package managers such as Cocoapods or Gradle, and other tools such as React Native, Xamarin, and PhoneGap. |

## 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 best operating platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments is a cross-platform framework. A cross-platform framework, such as React Native, allows the game to be developed once and then deployed on multiple operating systems, such as Windows, Mac, Linux, iOS, and Android. This eliminates the need to develop separate versions of the game for each operating system, saving time and development costs.
2. **Operating Systems Architectures**:   
     
   The chosen operating platform for Draw It or Lose It is a cross-platform framework, React Native. It allows the game to be developed once and deployed on multiple operating systems, and it's based on JavaScript, it follows a similar structure to web development, it allows for direct access to the native APIs of the device, which allows for more advanced features such as push notifications, camera access, and more. The game can be accessed through a web browser, and the game can be updated on the server-side, and users will automatically receive the updates the next time they access the game.
3. **Storage Management**:   
     
   An appropriate storage management system to be used with the recommended operating platform, React Native, is a cloud-based storage solution such as Amazon S3, Azure Blob Storage, or Google Cloud Storage. These solutions offer scalability, data durability, accessibility, and security features that are suitable for React Native applications.
4. **Memory Management**:  
     
   React Native uses memory management techniques such as virtual DOM, memory recycling, and garbage collection to optimize the performance of the Draw It or Lose It software. These techniques help to minimize the number of DOM updates required, reduce the amount of memory used by the application, and automatically free up memory that is no longer being used. Additionally, React Native also provides the ability to use third-party libraries for better memory management.
5. **Distributed Systems and Networks**:  
     
   One way this can be achieved is by using a client-server architecture. In this architecture, the game's server acts as the central point of communication between the different devices. The server is responsible for maintaining the state of the game, handling requests from clients, and sending updates to clients. The clients, which run on different devices, communicate with the server to request updates and send their own updates.
6. **Security**:

To protect user information on and between various platforms in Draw It or Lose It, several security measures can be implemented such as encryption, authentication and authorization, firewall, security updates, and network security. React Native allows for the use of third-party libraries to implement these security measures. It's important to keep the operating system and any third-party libraries that the game relies on up to date with the latest security patches and updates.