

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 | 09/19/2025 | Justin Vaughn | Completed the Executive Summary, Requirements, Design Constraint, and Domain Model sections. |
| 1.0 | 10/04/2025 | Justin Vaughn | Completed the Evaluation section. |
| 1.0 | 10/19/2025 | Justin Vaughn | Completed the Recommendations 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)

The Gaming Room has asked for our help in creating an online game for them titled Draw It or Lose It. They want their game to be web-based and available on multiple platforms, as it is only on the Android app store as of now. The business requirements for the game are to expand their reach with different platforms, ensure fairness and integrity in their game, and enhance the user experience. The technical requirements are having one or more teams in a game, each team will have multiple players assigned to it, game and team names must be unique, and only one instance of a game can exist at any given time.

## Requirements

The business requirements for the game are to expand their reach with different platforms, ensure fairness and integrity in their game, and enhance the user experience. The technical requirements are having one or more teams in a game, each team will have multiple players assigned to it, game and team names must be unique, and only one instance of a game can exist at any given time.

## [Design Constraints](#_2et92p0)

The game must be developed as a web-based, cross-platform application. This ensures that it can run across multiple platforms but also limits the development team to using tools that support multiple platforms.

The application must ensure that only one active game instance exists in memory at a time. This will help preserve fairness and prevent the duplication of resources.

The game must run smoothly across different platforms, which means the design must be lightweight and efficient code.

Game and team names must be unique. This influences the design of the code requiring name validation and iteration mechanisms.

The design of the application must allow for each game to include one or more teams, and each team to include multiple players. This requires efficient iteration when adding or checking teams and players.

## [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)

At the center of the UML diagram is the Entity class. This class acts as a base class that has shared attributes inside of it. By using inheritance, the Game, Team, and Player classes extend the Entity class. This helps eliminate code duplication and helps ensure consistency across the whole application.

The Game class has a list of Team objects which establishes a one-to-many relationship between the two classes. Similar to this, the Team class maintains a list of Player objects, which shows that each team can have multiple players. These things support the requirements that a game can have one or more teams, and each team can have multiple players.

The GameService class is responsible for managing a collection of games. It uses the Singleton design pattern to ensure only one instance of a game exists in memory at a time which helps fulfill the requirement that only one game instance can be active. The class also uses iteration when adding or retrieving games, which ensures that each game and team have unique names is fulfilled.

The UML diagram also shows key OOP principles. It shows encapsulation by having data fields kept private and allowing access through public methods. It shows inheritance by having Game, Team, and Player classes reuse common code found in the Entity class. It shows polymorphism by having methods such as toString() that can be overridden to provide class-specific output. Finally, it shows abstraction by having the Entity class use shared features of all domain objects.

**"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** | It is possible to host a web-based software application on macOS, however there are a lot of extra steps needed to accomplish it.  The advantages of macOS is that there is no additional license fee because it comes with the hardware, and the environment is very developer friendly.  The disadvantages are that Apple seems to be stopping support of their server applications. Another disadvantage is that even though there is no license fee to run a server on your own hardware, macOS is only legally allowed to run on Apple hardware which makes it a lot more expensive than other platforms. | Linux is the most popular server operating system in the world, being used by almost every major web service.  The characteristics of Linux is that its open-source and highly customizable, highly secure and stable, and supports a lot of modern web technologies.  The advantages are that it’s free to use, it has great performance and can support all the people that would play Draw it or Lose it, and it’s got a huge ecosystem of things that make life easier as a developer.  A couple of disadvantages is that it has a steeper learning curve than most other platforms, and it can take longer to get set up securely. | Windows has a service called Windows Server that allows you to host web applications.  The characteristics of the Windows servers is that it offers graphical tools so its easier for developers to use, it has a large range of languages that it supports, and it has a large ecosystem of tools and security.  The advantages are that its easy to use, it works seamlessly with other Microsoft products, and that its highly compatible.  The disadvantages are that it requires at the minimum a license for the Windows Server application and sometimes other licenses. It also uses a lot more resources than other platforms. | Though mobile devices can host lightweight servers, because they don’t have enough processing power, they are not used for large applications like Draw it or Lose it.  The characteristics of mobile devices are that they are designed as client devices for user interaction, and that they have limited storage and network bandwidth.  The advantages of them are that they are widely used as almost everyone has a mobile device of some kind, and they are great for testing applications on smaller screens and slower connections.  The disadvantages of them are that they heavily rely on mobile OS for user safety, and they are not designed to be powerful enough to host most things. |
| **Client Side** | The applications UI should be built as standard responsive web pages that work the same in each browser, and enough time to test each browser thoroughly should be set aside. A front-end developer would be needed to create the UI and bug test it. The costs for the application would mostly be for the developer, as there is no fee for an Apple license. | The Linux application creation process is like the Mac application process as the UI should also be built using standard responsive web pages. Enough time to test the layout, graphics, and features on each browser should be set aside. A front-end developer would be needed to create the responsive web pages and to test them. The costs would mostly be the developer, as there are no licensing fees for Linux. | The Windows process is like the previous platforms as it should also use standard responsive web pages that work across multiple browsers. Enough time should be set aside to test the application on each browser. A front-end developer would be needed to create the responsive pages and test them. The main cost would be the developer, as there is no licensing fee. | The mobile devices application should be built to adjust to smaller screens and with the ability to support touch controls. Time should be set aside to test the application on multiple different devices. A front-end developer familiar with mobile web design would be needed. The main cost would be the developer, as there are no licensing fees. |
| **Development Tools** | To build the front end of the application in macOS HTML5, CSS3, and JavaScript will be used, along with React to manage the UI. The development team will use Node.js to run the development server, and VS code can be used as the main IDE. Git and GitHub will be used for collaboration between the members of the development team. The testing for mac will be done using Docker Desktop.  Only one development team should be enough to create the application.  Almost everything used is free. The only thing that might cost money is if the development team deemed the premium version of anything is necessary. | Linux is almost the same process as MacOS. HTML5, CSS3, and JavaScript should be used for the front-end development. Node.js should be used to run the development server, while VS Code is the main IDE. Git and GitHub should be used for collaboration between the members of the development team. Docker should be used for testing.  Only one development team should be enough to create the application.  Almost everything is free unless premium is deemed as necessary. | Windows is almost the same process as the previous two. HTML5, CSS3, and JavaScript should be used for the front end, and React for the UI. Node.js should be used to run the development server, and VS Code will be the main IDE. Git and GitHub should be used for collaboration between the development team members. Testing should be done through Docker.  Only one development team should be enough to create the application.  Almost everything is free unless premium is deemed necessary. | The app will be built with HTML5, CSS3, and JavaScript using React for the UI. Node.js should be used to run the development server, and VS code should be used for the main IDE. Git and GitHub should be used for collaboration. Testing will require some extra steps for mobile devices. The team should use DevTools device emulation, Safari Responsive Design Mode, and Docker. These applications should help test touch input, scaling, and network connections.  One development team should be enough.  Because the app will run in the browser, there should be no fees unless premium is bought. |

## 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 deploying Draw it or Lose it onto a Linux-based server in the cloud. Then, using Docker to handle the Java REST backend. Running containers on a managed service provides portability and scaling. Linux easily supports expansion to multiple devices like Android, IOS, and desktop by exposing a single REST API.
2. **Operating Systems Architectures**: The Linux operating system uses a modular architecture that separates the kernel, system libraries, and user space. This structure provides strong stability, scalability, and security which are all important for a cloud-based game. In the structure, the kernel manages processes, memory, and hardware resources, while the system libraries allow the Java REST backend to interact with the system. This makes it easier to run the application in multiple environments.
3. **Storage Management**: I recommend using a cloud based relational database for Draw it or Lose it to store the smaller data like user accounts, scores, and game data. PostgreSQL works well with Linux and Docker. For larger assets like the images, an object storage service like Google Cloud Storage would work well. This set up for the storage allows for automatic scaling when needed, and high availability which are both important for keeping the game stable and responsive.
4. **Memory Management**: Linux uses an efficient virtual memory system that allows processes to use more memory than is physically available by swapping inactive data to disk. This will help Draw it or Lose it handle multiple active users and game sessions without slowing down. Linux also uses demand paging, which loads only the parts of a program that are currently needed. Within the application Java’s garbage collector automatically frees memory that is no longer needed.
5. **Distributed Systems and Networks**: For Draw it or Lose it to be able to communicate between multiple platforms, the application can use a distributed client server model connected through the internet. The server, which is hosted in the cloud, manages game sessions, user authentication, and data storage. Each client would communicate with the server using REST APIs for real time updates. Load balancers would be used to ensure traffic is split evenly between servers to prevent crashes. If a network outage does happen, the game can store data local temporarily and synchronize once the connection is restored.
6. **Security**: Draw it or Lose it should use HTTPS encryption to protect data sent between clients and the server. User authentication should be handled using secure tokens to prevent unauthorized access. On the server side, firewalls and regular system updates will help protect against attacks. Sensitive user information should be encrypted before being stored in the database.

Sources:

Corbo, A. (2025, April 9). *What is Linux? Definition, uses, advantages*. Built In. https://builtin.com/software-engineering-perspectives/linux