

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 11/19/2024 | Lasupe Xiong | Initial prototype software design |
| 1.1 | 12/03/2024 | Lasupe Xiong | Evaluation |
| 1.2 | 12/15/2024 | Lasupe Xiong | Recommendations |

**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 now provides an Android app called "Draw It or Lose It" and intends to create a web-based version that works across various platforms. This game, akin to the 1980s television show "Win, Lose, or Draw," consists of many games, each with various teams and participants. To work properly, each game and team name must be unique. The program offers a big library of stock drawings and consists of four rounds of play, each lasting one minute, with pictures generated slowly and completely at the 30-second mark. If a team fails to solve the puzzle in the allocated time, the other teams get 15 seconds to make one guess each. The new web-based platform will be available from a variety of computing contexts, including desktop and mobile devices. The team at The Gaming Room requires assistance in establishing this environment.

## Requirements

*To deliver a flawless gaming experience, the program must be able to manage a high number of concurrent users efficiently.*

*Real-time drawing interface: The application's interface should be responsive, allowing users to create and submit drawings in real time.*

*accurate evaluation and scoring: In order to effectively estimate the quality of user drawings, the application must examine and score them using precise criteria.*

*User registration and authentication: The application should provide a powerful user registration and authentication mechanism to manage user accounts and ensure safe access.*

*Multiplayer feature: The app should include a multiplayer option that allows users to compete against one another for a social and engaging experience.*

*Cross-platform accessibility: To reach a larger audience, the program should be developed to function across many computing environments, such as desktops and mobile devices.*

## [Design Constraints](#_2et92p0)

Building 'Draw It or Lose It' for Android, iOS, and the web requires traversing many SDKs while maintaining a consistent API across platforms. The game must enable teams on any device, with unique IDs for all elements (games, teams, and players) to prevent duplicates and enforce unique names (team leaders will be notified if a chosen name already exists). An evaluation class will evaluate and score user artwork using established criteria. Scalability is critical for supporting a growing user base and future needs. To provide excellent user experience, real-time sketching and assessment procedures must be rapid and fluid. Robust security protects user information by preventing unwanted access or data breaches. Consistency between devices and operating systems is critical. Design considerations include data security, regulatory compliance, optimal data processing and analysis, and selecting appropriate cloud services and deployment methods.

Object-oriented programming (OOP) techniques are ideal for these objectives. Encapsulation isolates data and functionality into classes, enabling modularity and code reuse. Inheritance enables us to reuse code and define a clear hierarchy for game sessions and assessments. Associations connect users, drawings, games, and assessments, allowing for data administration and interaction inside the system.

## [System Architecture View](#_ilbxbyevv6b6)

"Draw It or Lose Its’s system architecture is web-based and distributed. It consists of four major components: a web server, an application server, a database server, and client devices. The web server responds to incoming client requests and provides the application's web pages. The application server handles user interactions, controls game logic, and organizes multiplayer features. The database server holds user data, drawings, and game-related information. Client devices, such as PCs and mobile devices, communicate with the program via web browsers. This design guarantees efficient communication and data storage, allowing for seamless functionality across several computer environments.

## [Domain Model](#_8h2ehzxfam4o)

The provided UML class diagram demonstrates the core structure of the "Draw It or Lose It" application. It follows object-oriented ideas such as encapsulation, inheritance, and association. Each class demonstrates encapsulation by including its own properties (such as username in User) and methods (such as evaluate in Evaluation). Inheritance enables code reuse, as Game and Evaluation inherit properties and actions from their parent Entity class. Associations define relationships between items, allowing for data sharing.

**"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** | Macs may be used as servers, but licensing is costly, and development requires MacBooks. The advantages of Mac OS include trendily upgradeable hardware, a simple yet powerful GUI, flexible terminal commands for server configuration, and effective security. Apple has one of the greatest anti-malware applications to guard against spyware, adware, and worms, and utilizing a Mac as a server is simple. However, the expensive cost of Mac goods and license, as well as lower preference from web hosting services, are significant downsides. | Linux, which is ideal for web-based hosting, is the most popular and cost-effective choice due to its open license. Like Mac, Linux provides versatile terminal commands and great security, making it less vulnerable to cyber-attacks. Its open-source nature allows for complete customization, making it the best option for web hosting services. However, compared to Windows, Linux is less user-friendly and compatible, particularly for gaming. It is difficult to obtain software and drivers that meet web hosting requirements, and there are fewer pre-built Linux workstations available. | Windows servers are recognized for their security and ease of installation and usage, while licensing can be costly. Despite this, the operating system has some advantages, including ease of use, a large library of applications with solid support, and interoperability with other programs. However, Windows is vulnerable to viruses owing to frequent upgrades and may lack proper technical assistance. While the initial cost of licenses might be considerable, it is still cheaper than that of Apple goods, making it an attractive alternative for many customers. | While mobile devices can be utilized as servers, they lack the capacity to do high-end operations. However, they might be used for development reasons. Mobile phones have several advantages, including a wide user base, an easy-to-use interface, and a lesser vulnerability to infections such as malware. On the disadvantage, mobile hardware cannot be upgraded, and there is no assistance in the event of a malware attack. For maximum server performance, it is ideal to have a stationary server that can be tracked in a single area. |
| **Client Side** | Development for Mac involves a Mac Book because to its excellent and user-friendly SDKs, which increases the cost and necessitates experience with Swift coding. It requires modest knowledge and patience, but connection with Apple products is straightforward. Mac computers require less upgrades, which are generally automated, and excel at multitasking, with the ability to run macOS, Windows, and Linux software concurrently. However, the initial setup costs are substantial, and certain software may need monthly fees. Hardware changes are sporadic and can last several years. | Developing Linux involves much knowledge and time, as well as experience with Python. It is the most difficult to set up because of the scarcity of accessible software, yet it may be the most secure after Apple. Linux's open-source structure makes it the most affordable alternative. However, the lack of debugging tools results in more issues, and there is no technical help accessible. | Windows necessitates a high degree of knowledge, with the.NET framework suggested for security and functionality. However, it needs minimal knowledge and time overall. There is a vast range of Windows-based PCs available at different price points. Many programs are compatible, making it the ideal alternative for gaming while keeping costs comparable to Mac. However, forced upgrades can be time-consuming, and variable functionality and quality among off-brand manufacturers can be troublesome. Malware, spyware, and ransomware are all potential concerns to consider. | When designing mobile devices, it is critical to hire skilled app developers since user interaction and display methods differ from those used on the web. Both Android and Apple utilize proprietary languages, which can slow development and need more knowledge. Despite this, mobile development is both affordable and user-friendly, with faster page load times and better flexibility. Depending on the device and the frequency of updates, building and maintaining mobile applications can be pricey. |
| **Development Tools** | Swift will be used entirely through iCode while coding on a MacBook. The Mac is more suited for terminal operations than Windows and supports languages like HTML, CSS, and JavaScript. IDEs for Java, Python, PHP, and Ruby are available, as well as development tools such as PyCharm, Eclipse, Visual Studio, Notepad++, and a variety of online applications for the Mac OS environment. | Most Linux systems have Python by default, and you may code with IntelliJ's Ultimate IDE. Linux is compatible with comparable IDEs found in Mac OS, although it should be noted that all software is unsupported. | Visual Studio Code is the primary and preferred tool for developing Windows applications, with the majority of programs written in C++ or C#. Windows is simpler to use than Linux, yet it can run the same IDE. It is also feasible to run both Windows and Linux at the same time, giving you access to important software as well as a plethora of other development tools and resources. | For mobile development, you have three choices. To create for Android, you'll need someone who is proficient in Android Studio. For iPhones, you'll need a Mac Book user who is adept in Swift using iCode. Alternatively, you may use Unity, which uses C++ and can convert apps to Android or iPhone platforms, albeit a Mac is still required for iPhone conversions. You may also use technologies like Android and Swift to develop software that works across several platforms. |

**Server Side Evaluation**

|  |  |  |  |
| --- | --- | --- | --- |
| **platform** | **Server-Based Deployment** | **Characteristics & Advantages** | **Weaknesses** |
| **Mac** | Macs can be used as servers, but it’s not typical for large-scale web hosting. | Trendy, upgradeable hardware; simple GUI; strong security with anti-malware; effective terminal commands for server configuration. | High cost of Mac products and licenses; less common as server hosting choice; limited support from hosting services. |
| **Linux** | Ideal for server-based deployment, widely used for web hosting. | Open-source, cost-effective, highly customizable, excellent security, vast community support. | Less user-friendly than Windows; software and driver compatibility may be limited; fewer pre-built workstations. |
| **Windows** | Suitable for enterprise-level hosting, commonly used in corporate environments. | User-friendly, extensive software support, large application library, good interoperability. | High licensing costs, more prone to malware; require frequent updates and may lack robust support for advanced configurations. |
| **Mobile Devices** | Limited for server hosting; can be used for development purposes but not high-end operations. | Easy-to-use interface, large user base, generally secure. | Limited hardware upgrade options; lacks enterprise-level power for server roles. |

**Client Side Evaluation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Platform** | **Development Considerations** | **Software Compatibility** | **Expertise Requirements** |
| **Mac** | Requires MacBooks for development; excellent SDKs and development environment (e.g., Swift). | Fully supports web browsers and development tools. | Swift knowledge needed; expensive initial setup, some tools have monthly fees. |
| **Linux** | Requires extensive knowledge, including Python and terminal use. | Web browsers like Chrome and Firefox are supported; compatibility may require additional work. | High technical knowledge; lack of debugging tools can lead to more issues; fewer resources available. |
| **Windows** | High familiarity needed, especially for .NET and C#. | Good browser support and compatibility with web technologies. | Intermediate knowledge for web development; potential troubleshooting with variable PC quality. |
| **Mobile Devices** | Requires specialized knowledge in mobile development; user interface and interaction are different from web-based designs. | Supported by Safari (iOS) and Chrome (Android) browsers. | Advanced skills needed in both iOS (Swift) and Android (Kotlin/Java); can be costly to build and maintain for both platforms. |

**Development Tools and Languages**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Platform** | **Primary Languages** | **IDEs and Development Tools** | **Licensing Costs** | **Impact on Development Team** |
| **Mac** | Swift, HTML, CSS, JavaScript, Python, Ruby | Xcode, PyCharm, Eclipse, Visual Studio, Notepad++ | Free (Xcode); some tools have licensing fees. | Teams need Mac expertise, particularly for Swift; suitable for multi-platform testing. |
| **Linux** | Python, JavaScript, PHP, Ruby | IntelliJ, Eclipse, VS Code, PyCharm | Mostly free; open-source. | Teams may need familiarity with Linux-specific tools and command-line interface. |
| **Windows** | C++, C#, JavaScript, Python, HTML/CSS | Visual Studio, VS Code, IntelliJ | Free for some editions; paid versions available. | Development can be cross-platform; easy to adapt to different development environments. |
| **Mobile Devices** | Swift (iOS), Java/Kotlin (Android), JavaScript (React Native/Flutter) | Android Studio, Xcode, Unity | Free for most; iOS deployment needs a paid developer account. | Specialized teams needed; potentially high cost for development and maintenance, especially for native app support. |

**Evaluation final consideration**

* Mac: Ideal for development with high security and a streamlined environment, but costly.
* Linux: Best for cost-effective server hosting and open-source development; requires technical expertise.
* Windows: Suitable for broad application development; good for development teams already familiar with Microsoft tools.
* Mobile: More demanding in terms of specialized knowledge and development tools; requires higher costs for full support.

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

* **Operating Platform**: I suggest utilizing a Linux Ubuntu Server housed on a Kubernetes cloud architecture in order to offer the scalability and flexibility needed for Draw It or Lose It. This method offers interoperability with contemporary cloud orchestration technologies, strong community support, and affordable open-source alternatives. Redundancy and flexibility are ensured by utilizing multi-cloud systems like Google Cloud Platform, Microsoft Azure, and Amazon Web Services (AWS). By enabling load balancing and self-healing nodes, Kubernetes guarantees the game's services have low downtime and high availability.
* **Operating Systems Architectures**: A containerized system that makes use of tools like Docker and Kubernetes is the suggested architecture. This makes scalability, deployment simplicity, and service separation possible. Functionalities like user authentication, game logic, and real-time drawing services are best separated using microservices architecture, which enables separate updates and improvements without interfering with the operation of the entire system. This architecture is ideal for a distributed gaming application since it is based on the dependable Linux kernel and guarantees performance, security, and resource efficiency.
* **Storage Management**: I advise utilizing a hybrid storage solution that combines databases and distributed file systems to handle data effectively. AWS S3, Azure Blob Storage, or GCP Cloud Storage are scalable and dependable choices for storing files, including drawings and game materials. On the database side, MongoDB's NoSQL features can handle unstructured or dynamic data, including real-time chat logs and drawings, while PostgreSQL can handle structured data, such user profiles and game metadata. Access speed is improved, and a seamless user experience is guaranteed when SSD-based storage is used. A Kubernetes node may also maximize scalability and storage allocation as the user base expands.
* **Memory Management**: Draw It or Lose It's speed depends on efficient memory management. Rapid handling of huge datasets is ensured by the use of in-memory data processing frameworks such as Apache Spark. By keeping frequently requested data near the application's processing layer, distributed caching solutions like Redis or Memcached can lower latency. Additionally, the monitoring features of Kubernetes enable dynamic resource allocation according to system demand, guaranteeing optimal memory consumption. The virtual memory and garbage collection features of the Linux operating system further improve resource efficiency and stop memory leaks.
* **Distributed Systems and Networks**: It is advised to use distributed software architecture using WebSockets and RESTful APIs to facilitate communication across several platforms. Platform-neutral interactions are made possible via RESTful APIs, while real-time updates for multiplayer games are made possible using WebSockets. Reliability and security are improved by cloud-based networking technologies including load balancers, virtual private networks (VPNs), and content delivery networks (CDNs). During system maintenance or node failures, gaming remains uninterrupted because to Kubernetes' dynamic node management capabilities. Furthermore, fault-tolerance and redundancy techniques guarantee reliable connectivity and performance across platforms.
* **Security**: Strong security measures are necessary to safeguard private user information and provide safe platform-to-platform connectivity. Data confidentiality is guaranteed by end-to-end encryption with TLS for data in transit and AES for data at rest. To restrict access to sensitive data, a role-based access control (RBAC) system is implemented, which separates user roles (e.g., admin, player, team leader). Multi-factor authentication (MFA) and Identity and Access Management (IAM) protocols improve user authentication. Users' trust is increased when industry standards like the CCPA and GDPR are followed, which guarantee adherence to data protection laws.

The suggestions made fit the specifications needed to develop Draw It or Lose It into a high-performing, safe, and scalable platform. The Gaming Room can provide a smooth and captivating experience across several platforms by utilizing containerization and microservices, deploying a Linux-based Kubernetes infrastructure, and putting strong security and networking solutions in place.