;;

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 | 06/18/2024 | Tristen Bradney | Changes in 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 is seeking to transform their Android game, “Draw It or Lose It”, into a web-based game that is accessible from different platforms. The game includes teams that are guessing images that are rendered from a library of stock drawings. Our suggested solution involves a front-end developed with React.js, a backend developed with Node.js and Express.js, and MongoDB for data storage, allowing for individual team and game names. Kubernetes and Docker will be used for deployment. Key features include a game lobby for creating/joining games, a game play area displaying images and taking in guesses, and a scoreboard. Security will be managed with JWT and role-based access control. The project development will follow a structured timeline, using client feedback, to ensure that the game application is meeting The Gaming Room’s requirements.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

Developing the web-based version of “Draw It or Lose It” involves several design constraints that impact development. Simultaneous performance requires fast communication technologies, while scalability would require system architecture that is capable of horizontal scaling through cloud services and microservices. Consistency and state management in a distributed environment will rely on centralized state store or consistent hashing techniques. Security requires thorough authentication, encryption, and solid input validation. Cross-platform compatibility requires responsive design and extensive testing across devices and browsers. Data integrity involves ensuring that database systems are reliable and implementing regular backups. Unique team and game names need efficient database-level uniqueness safeguards. An engaging user interface requires intuitive design and regular testing. Efficient resource management will require orchestration tools, along with monitoring and scaling strategies. Lastly, rapid development cycles with CI/CD pipelines will support continuous integration and deployment, ensuring quick iterations and feedback incorporation. Addressing these constraints promotes the development of a thorough, scalable, and user-friendly application.

## [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 below displays the relationship between seven classes designed for the game, showing object-oriented principles. The ‘ProgramDriver’ class uses the ‘SingletonTester’ class, while ‘Entitiy’ serves as a base class for ‘Game’, ‘Team’, and ‘Player’, displaying inheritance and reusable code. The ‘GameService’ class uses the Singleton design pattern, making sure that game management is centralized and consistent. Associations indicate one-to-many relationships: ‘GameService’ to ‘Game’, ‘Game’ to ‘Team’, and ‘Team’ to ‘Player’, which supports scalability and reusable design. Encapsulation is displayed using private attributes and public methods, creating thorough and maintainable code. These principles and relationships create a hierarchical structure, efficient management of unique instances, and a scalable, maintainable, and organized codebase.

**"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** | Characteristics for Mac include strong integration with the Apple ecosystem, Mac is UNIX based and it has high performance. Advantages for Mac include high security, reliability and stability, and is also great for development, including iOS. Disadvantages include limited server hosting options compared to Linux and Windows and expensive hardware costs. For licensing costs, there is a high initial hardware cost, and no OS licensing costs for server use. | Characteristics of Linux are that it is free and open source. Linux also provides high stability and security along with strong server performance. Advantages include being widely customizable, cost effect, having a broad community for support and efficient resource management. Disadvantages include Less support for certain software and a steeper learning curve for beginners. There are no licensing costs for the OS. There could be possible costs for support versions. | Characteristics for Windows include extensive support for various software, Windows is widely used, and it has good performance and is beginner friendly and easy to use. Advantages for Windows include a familiar interface, very broad software compatibility and extensive support. Disadvantages include a higher vulnerability to malware, being resource intensive and licensing costs. For licensing costs, Windows has server licensing costs that can be potentially high for enterprise editions. | Characteristics for Mobile Devices include scalable backend services, can run containerized applications and efficient for handling REST APIs. Advantages include being cost-effective, high performance, a large community and support and flexible deployment options. Weaknesses include requiring skilled administrators and having compatibility issues with some proprietary software. For licensing costs, there are none for the OS and possible costs for enterprise support. |
| **Client Side** | Characteristics for Mac include HTML5, CSS and JavaScript for responsive web design. Browser compatibility: Chrome, Safari.  Advantages include having a single codebase for all platforms, a wide reach to users and no need for separate app installations. Disadvantages include browser inconsistencies and performance can vary by browser. Development considerations include the use of modern frameworks, like React, Angular or Vue, thorough testing across browsers and expertise in web development. | Characteristics for Linux include HTML5, CSS and JavaScript for responsive web design. Browser compatibility: Chrome, Firefox. Advantages include having a single codebase for all platforms, a wide reach to users and no need for separate app installations. Disadvantages include browser inconsistencies and performance can vary by browser. Development considerations include the use of modern frameworks, like React, Angular or Vue, thorough testing across browsers and expertise in web development. | Characteristics for Windows include HTML5, CSS and JavaScript for responsive web design. Browser compatibility: Edge, Chrome, Firefox. Advantages include having a single codebase for all platforms, a wide reach to users and no need for separate app installations. Disadvantages include browser inconsistencies and performance can vary by browser. Development considerations include the use of modern frameworks, like React, Angular or Vue, thorough testing across browsers and expertise in web development. | Characteristics for Mobile Devices include HTML5, CSS and JavaScript for responsive web design. Mobile browser compatibility: Safari for iOS, Chrome, Android. Advantages include having a single responsive web app for both platforms, and a lower development cost compared to native apps. Disadvantages include limited access to native device features and performance can be lower than native apps. Development considerations include the use of responsive frameworks, like Bootstrap, testing on multiple devices and screen sizes, and expertise in mobile web development. |
| **Development Tools** | Relevant programming languages include Swift (iOS) and JavaScript and development tools like Xcode, VS Code and Git. Impacts on the development team include Mac being excellent for iOS development and the possible need to balance between iOS-specific and cross-platform tools. For licensing costs, Xcode is free and other tools may have minimal costs. | Relevant programming languages include JavaScript, Python and Java and development tools like VS Code, Eclipse and Git. Impacts on the development team include flexibility in choosing tools, open-source options to reduce costs, and knowledge may be required for various languages and tools. Licensing costs are typically low or none and open-source tools are available. | Relevant programming languages include C#, JavaScript, .NET and tools like Visual Studio, VS Code and Git. Impacts on the development team include strong support for enterprise-level applications and requires expertise in Microsoft technologies. For licensing costs, Visual Studio has free and paid versions and other tools may have minimal costs. | Relevant programming languages include JavaScript, Swift, Kotlin and tools like VS Code, Android Studio, Xcode and Git. Impacts on the development team include the need for expertise in both web and native app development and cross-platform tools like React Native and Flutter can simplify development. For licensing costs, Android Studio and Xcode are free and other tools may have minimal costs. |

## 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 Gaming Room’s game, Draw It or Lose It, to effectively expand onto various computing environments, I recommend deploying the game on a Linux based server platform. Linux offers several advantages, including thorough security features, flexibility, and scalability, which are crucial to support a distributed game environment.
2. **Operating Systems Architectures**: There are several Linux operating system architectures that can be broken down into different components. The Kernel is the core component that manages the system resources like the CPU, memory, and devices. The System Libraries are essential libraries that provide an interface between the kernel and application software. The System Utilities are basic tools and applications required for system management. The User Interface can be a graphical user interface (GUI) or command-line interface (CLI) for user interactions. Linux supports various file systems (ext4, XFS, Btrfs), networking protocols, and offers containerization through Docker, which can contain the game environment, ensuring consistency across different deployments.
3. **Storage Management**: For storage management, I recommend using the Btrfs file system on the Linux platform. Btrfs is very efficient and has features like snapshotting, where you can quickly capture the state of the file system at any point in time, compression, that reduces the storage footprint, and copy-on-write (CoW), that enhances data integrity and performance. Additionally, Btrfs provides excellent support for large volumes of data and high scalability, which is ideal for the dynamic storage needs of an online game.
4. **Memory Management**: The Linux operating system uses several sophisticated memory management techniques to ensure optimal performance for applications like “Draw It or Lose It”. Linux manages memory through a virtual memory system, allowing each process to use more memory than physically available by swapping unused pages to a disk. Paging and swapping help manage memory usage by moving inactive pages to disk, ensuring active processes have the memory they need. Linux has efficient memory allocation and deallocation of memory blocks to processes using algorithms like buddy system and slab allocation. Frequently accessed data is stored in cache memory, significantly speeding up retrieval times. These techniques ensure that the game runs smoothly even under heavy load conditions.
5. **Distributed Systems and Networks**: To enable Draw It or Lose It to communicate effectively between various platforms, the game can leverage a microservices architecture deployed within a distributed system framework. The microservices architecture can break down the game functionality into smaller, independent services that can be developed, deployed, and scaled independently. Services can communicate over HTTP/HTTPS using RESTful APIs or message brokers like RabbitMQ or Kafka for real-time data exchange. Data consistency ensures consistency across distributed components using distributed databases like MongoDB or Cassandra. Utilizing networking protocols and tools such as Kubernetes for container orchestration and load balancing to manage traffic and ensure high availability. Also, implementing strategies for handling network outages and ensuring connectivity, such as redundant network paths and automated failover mechanisms.
6. **Security**: To protect user information and ensure security across various platforms, several measures should be implemented. Use TLS/SSL for encrypting data in transit and AES for data at rest. Implement strong authentication mechanisms (OAuth2, JWT) and role-based access control (RBAC) to manage user permissions. Regularly update the system and use security protocols to protect against invulnerability. Deploy firewalls and intrusion detection/prevention systems to monitor and protect the network. Use checksums and hash functions to ensure data integrity during storage and transmission. Conduct periodic security audits and vulnerability assessments to identify and mitigate potential security risks.