

<Name-of-Software-Application>

# **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 | <mm/dd/yy> | <Your-Name> | <Brief description of changes in this revision> |

**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 software design problem at hand involves creating a web-based gaming application called "Draw It or Lose It." This application is inspired by the 1980s television game "Win, Lose or Draw," where teams compete to guess phrases, titles, or things represented through drawings. In our digital version, teams will guess phrases based on rendered stock drawings instead of a player drawing them.

## Requirements

*To address the client's requirements, we propose the development of a robust and user-friendly web-based gaming application. This application will allow for one or more teams, each consisting of multiple players. Unique game and team names will be enforced to prevent naming conflicts. Additionally, we will ensure that only one instance of the game can exist in memory at any given time by generating unique identifiers for games, teams, and players*

## [Design Constraints](#_2et92p0)

Developing the "Draw It or Lose It" game application in a web-based distributed environment comes with specific design constraints:

Web-Based Environment: The application must be accessible through web browsers, making it platform-independent and easy to reach for a wide audience.

Scalability: The application should handle multiple concurrent games and teams. We must design it to scale efficiently as the user base grows.

Data Consistency: Ensuring data consistency among distributed components is crucial. Implementing appropriate data synchronization mechanisms is essential to avoid conflicts.

Security: Protecting user data, including team and game information, is paramount. We must implement user authentication and authorization to safeguard sensitive data.

Performance: Rendering stock drawings and managing real-time interactions require optimized performance. We need to minimize latency to provide a seamless gaming experience.

Unique Identifiers: To prevent memory issues, we will generate and manage unique identifiers for games, teams, and players. This will ensure that only one instance of the game exists at any time.

User Experience: The user interface (UI) should be intuitive, responsive, and engaging. Players should be able to join and interact with their teams easily.

Testing and Debugging: Extensive testing, including unit testing, integration testing, and user testing, will be necessary to identify and resolve issues.

## [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)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

**"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 computers, with their Unix-based OS and strong security features, provide a stable and secure environment for hosting web-based software applications. However, their limited hardware choices and higher price point compared to PCs can impact scalability and cost-effectiveness in larger-scale hosting scenarios. | Linux servers excel in hosting web-based software applications with their open-source flexibility, robustness, and high customizability. They are renowned for their security and stability, making them a top choice for hosting. However, Linux can be less user-friendly for beginners, and some proprietary software may not be readily available or compatible, demanding more technical expertise to set up and maintain. | Windows servers offer a user-friendly interface and broad software compatibility, including Microsoft products, but may be more susceptible to security vulnerabilities. They tend to be costlier due to licensing and resource requirements, and they are less commonly used for web hosting compared to Linux, which is favored for its stability and security. | Hosting web-based software on mobile devices is constrained by their limited computational power and memory. While mobile devices offer portability and accessibility, they pose challenges due to performance limitations and varying network conditions. Successful hosting requires optimizing applications for mobile platforms. |
| **Client Side** | Supporting multiple client types on Mac necessitates expertise in macOS development, robust testing, and compliance with Apple's guidelines. Consider costs for Mac hardware, development tools, and app distribution. Allocate resources for maintenance, updates, user support, and localization, ensuring compliance with security and privacy. Additionally, plan for marketing and iterative improvements based on user feedback and competitive analysis. | Supporting multiple client types on Linux requires a skilled Linux development team, rigorous testing across diverse Linux distributions, and adherence to Linux conventions. Account for expenses related to Linux environments and tools. Prepare for ongoing maintenance, updates, user support, and potential localization efforts. Ensure security and privacy compliance. Additionally, allocate resources for marketing and iterative improvements based on user feedback and competitive analysis. | Supporting multiple client types on Windows necessitates expertise in Windows development, thorough testing across different Windows versions, and compliance with Microsoft's guidelines. Consider costs for Windows-based development tools and environments. Prepare for continuous maintenance, updates, user support, and potential localization. Ensure adherence to security and privacy standards. Allocate resources for marketing and iterative improvements based on user feedback and competitive analysis. | Supporting multiple client types on mobile platforms, such as iOS and Android, demands specialized mobile development expertise, rigorous testing across various devices and OS versions, and adherence to platform-specific guidelines (e.g., Apple's HIG and Google's Material Design). Budget for development tools, testing devices, and potential app store fees. Plan for ongoing maintenance, updates, user support, and localization. Prioritize security, privacy compliance, and efficient resource management. Allocate resources for marketing, user feedback integration, and iterative improvements based on competitive analysis. |
| **Development Tools** | To develop and deploy software on Mac, developers typically use programming languages such as Swift and Objective-C. The official integrated development environment (IDE) is Xcode, which offers a code editor, debugging tools, and Interface Builder for designing user interfaces. Version control is managed using Git, while package management is facilitated by Homebrew. Designers often use Sketch for UI/UX design, and testing relies on XCTest and Instruments. Deployment and distribution can be handled through Mac App Store Connect, with notarization tools ensuring app security. Continuous integration and continuous deployment (CI/CD) tools like Jenkins or Travis CI automate the build and release processes. Localization tools within Xcode assist with translating apps into different languages. | Linux software development often employs languages like C, C++, or Python, with various text editors or IDEs available, such as Visual Studio Code or CLion. Git is essential for version control, and package management varies by distribution, including APT, YUM, or Snap packages. UI/UX design tools like Inkscape may be used, with testing frameworks like GTest for C++ applications. Deployment on Linux typically involves package creation (e.g., DEB or RPM) and repositories. CI/CD can be configured using Jenkins or GitLab CI. Localization tools are used for translating software into different languages. | Developing and deploying software on Windows involves languages like C# and C++, with Visual Studio as the primary IDE, offering code editing, debugging, and testing capabilities. Git is commonly used for version control, and Windows Package Manager (winget) simplifies software installation. Design tools like Adobe XD or Figma may be used for UI/UX design, while testing can involve Microsoft's testing frameworks. Deployment can be done through the Microsoft Store, and CI/CD pipelines can be set up using Azure DevOps or Jenkins. Localization tools help adapt software for international markets. | Developing mobile apps for iOS typically involves using Swift or Objective-C in Xcode, while Android development relies on Java or Kotlin in Android Studio. Version control with Git, package management using CocoaPods (iOS) or Gradle (Android), and UI/UX design with tools like Adobe XD or Sketch are common practices. Testing frameworks such as XCTest (iOS) or Espresso (Android) ensure app quality. Deployment occurs through the Apple App Store (iOS) or Google Play Store (Android), with CI/CD pipelines set up via Jenkins or GitHub Actions. Localization tools help translate apps into various languages to reach a broader audience. |

## 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**: Java can run on various operating systems, including Windows, Linux, and macOS. Therefore, you can maintain the use of Docker and Kubernetes for containerization and orchestration, as these technologies support Java applications effectively.
2. **Operating Systems Architectures**: Java is platform-independent by design. You can package your Java application in a Docker container that uses a base image compatible with the desired operating systems.
3. **Storage Management**: Continue to use a combination of cloud-based and relational databases, as mentioned earlier. Java offers robust libraries and frameworks for interacting with different database systems, making it versatile for storage management.
4. **Memory Management**: Java has its memory management mechanism with the Java Virtual Machine (JVM). The JVM handles memory allocation and garbage collection, ensuring efficient resource utilization within Java applications. You can configure JVM memory settings to optimize the Draw It or Lose It software.
5. **Distributed Systems and Networks**: Java provides extensive support for building distributed systems. You can use Java technologies like Java RMI (Remote Method Invocation) or implement RESTful APIs with Java frameworks such as Spring Boot for communication between various platforms. These Java-based approaches will work well with Docker and Kubernetes.
6. **Security**: For security in the Java-based Draw It or Lose It application, employ robust user authentication and authorization mechanisms using Java libraries or frameworks, such as Spring Security. Implement secure cross-platform communication with HTTPS, regularly update Java runtimes and containers, configure security settings within Docker and Kubernetes, enforce network policies, firewalls, and data encryption to protect user information and ensure a secure gaming experience.