

The Gaming Room

# **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 4**](#_Toc115077322)

[**System Architecture View 6**](#_Toc115077323)

[**Domain Model 7**](#_Toc115077324)

[**Evaluation 9**](#_Toc115077325)

[**Recommendations 10**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.1 | 09/30/2023 | David Greene | Updated Evaluation section |
| 1.2 | 10/15/2023 | David Greene | Updated Requirements Section |

## [Executive Summary](#_sbfa50wo7nsh)

The software design problem is to develop a web-based version of the game application "Draw It or Lose It." The goal is to create a game where teams compete to guess what is being drawn from a library of stock drawings. The game consists of four rounds, each lasting one minute, with drawings rendered at a steady rate. If a team fails to guess the puzzle within the time limit, other teams have a chance to offer a guess within a shorter time frame.

To address this problem, we propose developing a web application that provides a user-friendly interface for teams to participate in the game. The application will render stock drawings as clues and allow teams to submit their guesses within the given time limits. The game will be designed to be engaging, intuitive, and enjoyable for players.

## Requirements

Business Requirements:

1. Multiple Platform Support: The game should be developed to serve multiple platforms, including web browsers, to reach a wider audience.

2. Game Concept: The game should be loosely like the television game Win, Lose or Draw, where teams compete to guess what is being drawn.

3. Stock Drawing Library: The game should utilize a large library of stock drawings as clues for the game rounds.

4. Game Rounds: Each game should consist of four rounds, with drawings rendered at a steady rate and fully complete at the 30-second mark.

5. Guessing Opportunity: If the team does not guess the puzzle before time expires, the remaining teams should have an opportunity to offer one guess each to solve the puzzle within a 15-second time limit.

Technical Requirements:

1. Team and Player Management: The game should support the ability to have one or more teams involved, with multiple players assigned to each team.

2. Unique Names: Game and team names should be unique to avoid conflicts and allow users to check name availability when choosing a team name.

3. Unique Identifiers: Unique identifiers should be generated for each instance of a game, team, and player to ensure only one instance of the game exists in memory at any given time.

## [Design Constraints](#_2et92p0)

1. Web-based Distributed Environment: The game application will be developed for a web-based distributed environment. This means that the application will be accessible through web browsers and will need to handle multiple concurrent users.

Implications: The application will need to be designed with scalability and performance in mind to handle a potentially large number of users simultaneously. The use of technologies such as cloud hosting and load balancing may be necessary to ensure smooth gameplay.

2. Time Constraints: The game consists of four rounds, each lasting one minute, with drawings fully complete at the 30-second mark. Teams have an additional 15 seconds to offer a guess if the puzzle is not solved within the time limit.

Implications: The application must accurately track and enforce the time limits for each round. It should provide visual cues to indicate the remaining time and notify teams when the time is up. Additionally, the application should handle the transition between rounds smoothly.

3. Stock Drawing Library: The application will utilize a large library of stock drawings as clues for the game.

Implications: The application needs to efficiently store and retrieve drawings from the library. It should provide a mechanism to select and render the appropriate drawing for each round. Consideration should be given to organizing and categorizing the drawings for easy retrieval.

4. User Interface and Experience: The application should provide an intuitive and user-friendly interface for teams to participate in the game. It should be visually appealing and responsive to user interactions.

Implications: The application needs to be designed with a focus on usability and accessibility. Clear instructions and prompts should be provided to guide teams through the gameplay. The interface should be responsive and provide real-time feedback to enhance the user experience.

[System Architecture View](#_ilbxbyevv6b6) (Possible future implementation:)

The following is a system and subsystem architecture for a web-based game application developed in Java:

System architecture:

The system architecture consists of the following tiers:

* Presentation tier: This tier is responsible for displaying the game interface to users and handling user input. It is typically implemented using HTML, CSS, and JavaScript.
* Application tier: This tier contains the game logic, such as tracking the game state, handling player actions, and generating game events. It is typically implemented using a Java programming language such as Java EE or Spring Boot.
* Data tier: This tier stores the game data, such as player profiles, game scores, and game settings. It is typically implemented using a relational database management system (RDBMS) such as MySQL or PostgreSQL.

Subsystem architecture:

The subsystem architecture consists of the following subsystems:

* Game subsystem: This subsystem is responsible for managing the game state, handling player actions, and generating game events.
* Player subsystem: This subsystem is responsible for managing player profiles and tracking player progress.
* Team subsystem: This subsystem is responsible for managing teams and tracking team scores.
* Storage subsystem: This subsystem is responsible for storing and retrieving game data.

Physical components:

The physical components required for the web-based game application will depend on the specific needs of the project. However, some typical components include:

* Web servers: To host the presentation tier and application tier.
* Database servers: To host the data tier.
* Load balancers: To distribute traffic between multiple instances of the web servers.
* Firewalls: To protect the application from unauthorized access.

Logical topology:

The following logical topology shows how the communication and storage aspects of the web-based game application are organized:

The presentation tier communicates with the application tier using HTTP requests. The application tier communicates with the data tier using JDBC or JPA. The presentation tier and application tier can be deployed on the same physical servers or on separate servers. The database tier is typically deployed on separate servers.

## [Domain Model](#_8h2ehzxfam4o)

**A screenshot of a computer

Description automatically generated**

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **A white apple with a bite taken out of it  Description automatically generatedMac** | **Linux**  **A penguin toy with yellow feet  Description automatically generated** | **Windows**  **A blue square windows with black background  Description automatically generated with medium confidence** | **Mobile Devices**  **A white square with blue and green letters  Description automatically generatedA green robot with white border  Description automatically generated** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac servers are known for their stability, security, and ease of use. They are commonly used for hosting web-based applications. However, it's important to consider the hardware and software requirements, as well as the cost and expertise needed for maintenance and support. | Linux servers are popular for web hosting due to their stability, security, and flexibility. They offer a wide range of open-source tools and technologies. However, Linux may require more technical expertise for setup and configuration compared to other platforms. | Windows servers provide a familiar environment for developers and offer good compatibility with Microsoft technologies. They are widely used for hosting web applications. However, licensing costs and potential security vulnerabilities should be considered. | Firstly, their portability allows users to access applications on the go, providing convenience and flexibility. Additionally, the touchscreen interface of mobile devices offers intuitive and interactive user experiences, enhancing usability. Weaknesses include:  Weaknesses include:  Limited Screen Size  Performance  Platform Fragmentation  Battery Life |
| **Client Side** | Developing software for Mac clients may require expertise in macOS-specific technologies and frameworks. The cost and time required for development may vary depending on the complexity of the application and the availability of development tools. | Supporting multiple types of clients on Linux may require considering cross-platform compatibility and using open-source technologies. The cost and time required for development may depend on the specific requirements of the application and the availability of development tools.to Linux | Developing software for Windows clients may involve using Microsoft technologies and frameworks. The cost and time required for development may vary depending on the complexity of the application and the availability of development tools. | Developing software for mobile devices requires expertise in mobile app development frameworks and languages such as Java or Kotlin for Android, and Swift or Objective-C for iOS. Considerations such as platform-specific design guidelines, performance optimization, and device compatibility should be considered. |
| **Development Tools** | Relevant programming languages and tools for Mac development include Xcode (IDE) for iOS and macOS development, Swift and Objective-C for iOS/macOS apps, and frameworks like SwiftUI and Cocoa. | Relevant programming languages and tools for Linux development include languages like Python, Java, C++, and tools like Eclipse, IntelliJ IDEA, and Visual Studio Code. | Relevant programming languages and tools for Windows development include languages like C#, .NET framework, Visual Studio IDE, and frameworks like ASP.NET for web development. | Relevant programming languages and tools for mobile app development include Java or Kotlin for Android development using Android Studio IDE, and Swift or Objective-C for iOS development using Xcode IDE. |
| **Licensing Considerations** | Hosting a web application on macOS typically involves using macOS Server, which may have licensing costs associated with it. The specific licensing costs would depend on the version and edition of macOS Server chosen. | Linux distributions are generally open-source and free to use. However, some enterprise-grade Linux distributions may offer additional support and services that come with licensing costs. The specific licensing costs would depend on the chosen distribution and the level of support required. | Hosting a web application on Windows can be done using Windows Server, which also has licensing costs. The licensing costs for Windows Server vary depending on the edition and the number of users or processors. | For mobile platforms like iOS and Android, there are no direct licensing costs associated with hosting web applications. However, if you plan to distribute the application through app stores like the Apple App Store or Google Play Store, there may be fees involved in becoming a registered developer and submitting the application for distribution |

Recommendations

**Recommendations for the web-based game development software requirements:**

In this section, we will discuss the recommended technologies and approaches for developing a mobile app that meets the specified requirements. Our primary goal is to ensure the app's compatibility, efficiency, scalability, and security, while catering to the need for multiple platform support, a stock drawing library, memory management, distributed systems, and network connectivity.

**Operating Platform: Linux (Ubuntu)**

To provide a seamless experience across multiple platforms, we recommend Linux as the operating platform, specifically the Ubuntu distribution. As an open-source operating system, Ubuntu offers robust compatibility with web-based applications, allowing for wide-ranging accessibility and ease of use.

**Operating System Architectures: Microservices architecture using Node.js**

To achieve modularity and scalability, we propose adopting a microservices architecture for the mobile app. By utilizing Node.js, a JavaScript runtime environment, we can easily develop and manage independent microservices that operate harmoniously across various platforms. Node.js enables efficient event-driven programming, making it an ideal choice for building distributed systems and ensuring seamless integration.

**Storage Management: Amazon S3 for file storage, MySQL for relational database management**

For the extensive stock drawing library, we recommend employing Amazon S3, a scalable and durable object storage service. This allows for secure storage of drawings, ensuring quick and reliable access for app users. In addition, we suggest using MySQL as the relational database management system to efficiently manage and query data associated with each drawing.

**Memory Management: Java with Ehcache for caching**

Efficient memory management is crucial for optimal performance. Therefore, we recommend utilizing Java, a versatile and popular programming language, which provides built-in memory management features such as garbage collection and dynamic memory allocation. To further enhance memory efficiency and improve performance, integrating Ehcache, a renowned caching library, can leverage caching strategies for frequently accessed data.

Distributed Systems and Networks: RESTful API using Node.js and Express.js

For seamless communication and scalability, we propose implementing a RESTful API using Node.js and the Express.js framework. This combination enables the development of robust and efficient APIs, facilitating easy integration across different components or systems.

**Security: Spring Security for Java-based applications**

To ensure the security of user data and prevent unauthorized access, we recommend utilizing Spring Security, a widely adopted framework for Java-based applications. Spring Security provides a comprehensive set of features, including secure authentication, role-based access control, and protection against common security vulnerabilities.

Conclusion:

In conclusion, the recommended technologies for building the mobile app include Linux as the operating platform, microservices architecture using Node.js, Amazon S3 for storage management, Java with Ehcache for memory management, a RESTful API with Node.js and Express.js for distributed systems and networks, and Spring Security for ensuring robust security. These technologies address the specified requirements, while considering compatibility, efficiency, scalability, and security. By adopting these recommendations, we can pave the way for a successful and user-friendly mobile app.

**Specific recommendations for each software requirement:**

Java program:

Requirement 1: A game will have the ability to have one or more teams involved.

* Create a Game class with a List<Team> property.
* The Game class can also have a CurrentTeam property to track the team that is currently playing.
* Create a Team class with a Name property to identify the team.
* Create a List<Team> to store all of the teams in the game.
* Implement methods to add a team to the game, remove a team from the game, get the current team, and set the current team.

Requirement 2: Each team will have multiple players assigned to it.

* Create a Player class with a Name property to identify the player.
* Implement methods to add a player to a team and remove a player from a team.
* Get the list of players on a team.

Requirement 3: Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.

* Create a database table to store game and team names.
* Apply a unique constraint on the Name column in the database table.
* Implement a method to check if a game or team name is already in use.

Requirement 4: Only one instance of the game can exist in memory at any given time.

* Use a distributed cache such as Redis or Memcached.
* Generate a unique identifier for each game when it is created and store it in the cache.
* Store the unique identifiers of the teams and players in the cache as well.
* Remove the game's unique identifier from the cache when the game is ended.

**Additional considerations:**

* Scalability: Consider using a cloud platform such as GCP, AWS, or Azure to scale the game application, allowing it to handle increasing numbers of users and data.
* High availability: Utilize a load balancer to distribute traffic between multiple instances of the game application, ensuring better performance and availability.
* Security: Add authentication and authorization mechanisms to protect user data and prevent unauthorized access and cheating within the game.

By incorporating these specific recommendations into the Java program, you can fulfill the requirements while prioritizing efficient team and player management, unique game and team names, memory management, and ensuring only one instance of the game exists in memory at a given time. Additionally, considering scalability, high availability, and security measures will enhance the overall performance and user experience of your game application.

Conclusion:

The above recommendations provide a starting point for developing the web-based game application being discussed the specific technologies that are chosen will depend on the specific needs of the project and the budget.