

<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 | <11/12/23> | <Collin Zielinski> | <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 objective is to develop a web-based version of “Draw It or Lose It” that serves multiple platforms. The goal is to stay close to the original Android app while incorporating web technology and different operating systems. The game will include teams and players, and the use of Singletons and Iterators.

## Requirements

The game must be accessible on different devices and operating systems, including PC and mobile. The game should allow multiple teams to participate simultaneously. Each game, team, and player must have an identifier. One instance of the game should exist at once. It should keep the engaging nature of the Android application.

## [Design Constraints](#_2et92p0)

A constraint is ensuring that the game functions uniformly across different web browsers and operating systems (Windows, macOS, Linux, Android). Unique identifiers must be made for games, teams, and players. Managing a single game instance in memory is also necessary and ensuring stable and working gameplay.

## [System Architecture View](#_ilbxbyevv6b6)

Client Tier, Server Tier, and Database Tier. Communication and storage must also be handled. The game must also have logic, user management, and networking system.

## [Domain Model](#_8h2ehzxfam4o)

The entity class will act as a base class with common attributes like ‘id’ and ‘name’, used by all other classes. The Game, Team, and Player classes are inherited from the entity class, each with specific attributes retaining to the class. The relationships between the classes reflect real world interactions.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Good security and stability, usually more expensive and has a smaller market share. | Offers flexibility and is used in server environments. Is open source but requires more technical expertise. | Most popular, user friendly, more open to security threats. | Has portability and accessibility, has hardware limitations. |
| **Client Side** | Costlier ecosystem requires compatibility with macOS. Must be specifically made for macOS. | Open source, so the cost will be reduced. Diverse distributions can increase client-side testing efforts. | Broad user base means an easier production, development tools are widely available. | More expansive development due to complexity of OS. Will be more expensive and time consuming. |
| **Development Tools** | Swift and Objective-C for applications; cross-platform tools like Flutter are of great help. | Popular languages include Python, Java, and C++. Eclipse can be used or Visual Studio Code. | .NET framework is commonly used, Visual Studio is a great IDE used in past project. | Kotlin for Android is popular, Swift for iOS. |

## 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**: Cross platform and web-based, possibly using technologies like HTML5, CSS, and JavaScript. This ensures compatibility across devices and operating systems. A consistent user experience and interface is necessary and of high quality. Using web technologies that make updates and maintenance easy will make sure users are all updated at the same time.
2. **Operating Systems Architectures**: A server-client model where the server handles game logic but the client interfaces with the user. The server will handle game logic, data processing, and central storage. It will also hold images if users choose to do so. The client will handle interactions and user interface. This separation allows for efficient resource allocation. Also, a server within our company is much more powerful than the average consumer pc, allowing for the game to run smoothly.
3. **Storage Management**: Cloud-based solutions for all players will make gameplay smoother. As mentioned in the operating architecture, robust memory management is necessary. Our large and expensive servers can handle most of image processing along with user login information to be accessed globally. This also makes data synchronized and backed up within our systems.
4. **Memory Management**: Something that can efficiently allocate, and garbage collect. Chosen by what is previously chosen. If played in web, JavaScript engines in browsers can handle memory management. This allows for managing of memory for people with weaker hardware.
5. **Distributed Systems and Networks**: Something for real time communication, considering network latency and disconnection. Real time communication will allow for seamless interactions between clients and the server. The gameplay needs to be synchronized, and if network connection is lost needs to be saved and re-accessible easily.
6. **Security**: Implement HTTPS, data encryption, and secure authentication methods to protect server data. Good data encryption will protect important end expensive user data resting in the server and in transit. A good username/password management system needs to be put in place. Regular security audits and checks can assure nothing has gone wrong or leaked.