

<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 | <11/26/2023> | <Dylan Schwering> | <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)

Creative Technology Solutions (CTS) is ready to change their clients game, “Draw It or Lose It” from an Android app into a web-based game. The goal is to expand the game’s reach across multiple platforms. Along with this, the new web platform needs to have expanded capabilities, which include Multi-Team capabilities, unique identifies for game sessions, teams, and player names, and finally enforced single instance of the game. These improvements are designed to increase the user experience and the user base.

## Requirements

The Gaming Room has set the following requirements

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

* The application must perform consistently across various web browsers and devices.
* Real-time gaming requires prompt data transmission.
* As only one instance of the game can exist in memory at a time, we will implement a state management system that prevents duplicate instances, possibly using a Singleton pattern.
* We need a reliable system to generate and manage unique identifiers for games, teams, and players.

## [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.>

* ProgramDriver: Entry point of the application, has the main() method. Uses the SingletonTester class
* SingletonTester: Has method testSingleton(). Testing the singleton design pattern, ensuring only one instance of the class.
* GameService: This class manages the game. It contains the list of game objects and methods for adding games, getting games, game counts, and player/team IDs. This class implements the singleton pattern through the getInstance() method.
* Game: This class represents the game, and has a list of Team objects. It extends the Entity class, therefore inheriting it’s methods and variables. It has an ID, name, and methods to manage teams.  
  Team: This class represents a team in the game. It also extends the Entity class and has a list of Player objects. It includes methods to add a player and return a string of the team,
* Player: This class represents a player and is the last class that extends the Entity class. This class includes a method to return a string as the player name.
* Entity: This is the base class the provides the common variables, ID and name, as well as the methods getId(), getName() and toString().

Along with Game, Team, and Player all extending the Entity class, Game, Team, and Player have zero-to-many relationships going in that order.

The OOP principles demonstrated are Encapsulation (each class encapsulates its data), Inheritance, and Abstraction (The Entity class provides and abstraction layer for the other three)

**"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 offer a secure and stable Unix-based environment for web apps but come with higher costs and limited scalability options compared to other servers | Linux is preferred for its flexibility and cost-effectiveness, though it requires more technical expertise to manage and configure. | Windows servers integrate well with Microsoft products, offering a user-friendly interface at the expense of higher costs and perceived security concerns. | Mobile devices are not suited for hosting applications due to limited hardware capabilities and network stability issues. |
| **Client Side** | Supporting Mac clients requires development tools compatible with macOS, often at a higher cost and with an expectation of polished user interfaces. | Linux client support is cost-effective with open-source tools but can be complex due to the variety of distributions and user expertise levels. | Windows has a large user base and requires compatibility across versions, with development potentially involving licensing costs for tools like Visual Studio. | Mobile development demands a different UI/UX approach, with the necessity for expertise in platform-specific SDKs and potentially higher testing needs. |
| **Development Tools** | Mac development typically involves Xcode with languages like Swift, along with support for other popular web development tools and languages. | Linux supports diverse programming languages and open-source IDEs, offering extensive customization for project needs. | Development on Windows often centers around Visual Studio and languages like C#, with ASP.NET for web applications on Windows servers. | Mobile app development requires platform-specific SDKs such as Android SDK or iOS SDK, with languages like Kotlin for Android and 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**: A web-based solution running on a Linux server is recommended for maximum scalability and flexibility, allowing "Draw It or Lose It" to be accessible from any device with a web browser.
2. **Operating Systems Architectures**: Linux provides a secure and stable environment with various architectures, supporting scalable server solutions with its monolithic kernel design.
3. **Storage Management**: PostgreSQL for transactional data management, combined with Amazon S3 for scalable static content storage, is suggested for the Linux platform.
4. **Memory Management**: Linux employs virtual memory and demand paging, optimizing resource allocation for "Draw It or Lose It" while ensuring process isolation and security.
5. **Distributed Systems and Networks**: "Draw It or Lose It" should use RESTful APIs for cross-platform communication, with microservices and load balancing for reliability against network outages.
6. **Security**: Implement TLS/SSL, OAuth, and data encryption to secure user information, with Linux's SELinux and firewalls providing additional layers of security.