

Stephen Bailey

# **CS 230 Project Software Design Template**

Version 2.0

## 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 | 07/17/2024 | Stephen Bailey | Initial version of the software design document |
| 2.0 | 07/31/2024 | Stephen Bailey | Added evaluation of operating platforms |
| 3.0 | 08/12/2024 | Stephen Bailey | Added recommendations for system architecture, storage, memory, distributed systems, networks, and security |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room looks to expand its current Android based game, “Draw It or Loose It”,into a web-based application that supports multiple platforms. This document outlines the software design plan to achieve this. The solution involves creating a scalable, maintainable, and efficient gaming application. This gaming application will be capable of managing multiple teams with unique name and ensuring a single game instance in memory. The key requirements include unique identifiers for each game, team, player, and seamless user experience across various devices.

## Requirements

*The client has made the following business and technical requirements.*

* *Business Requirements:*
  + *The game should support multiple teams.*
  + *Each team can have multiple players.*
  + *Team and Game names must be unique.*
  + *Only one instance of the game should exist in the memory at any given time period.*
* *Technical Requirements:*
  + *Unique identifiers for each team, player, and game.*
  + *The game must be web-based with the ability to be accessed from multiple platforms.*
  + *The application must ensure data integrity and handle multiple users effectively.*

## [Design Constraints](#_2et92p0)

The development of the game application in a web-based distributed environment presents multiple design challenges.

* Scalability – The application must be able to handle multiple users and teams without encountering performance issues.
* Maintainability – The codebase should be modular and adhere to best practices to accommodate easy updates and future additions.
* Data integrity – Unique names and identifiers are crucial to prevent conflicts as well as maintaining a smooth game play experience for the user.
* Concurrency - The system must handle simultaneous gaming sessions and real time updates without encountering data inconsistencies.
* User Experience – The application must provide seamless experience across a variety of devices and platforms.

These constraints will guide the selection of technologies, design patterns, architecture, ensuring the application meets the clients standard.

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram for the game application includes the following classes:

* **Entity:** base class holding common attributes id and name.
* **Game:** inherits from Entity class, representing a game instance.
* **Team:** inherits from Entity class, representing a team within the game.
* **Player:** Inherits from Entity class, representing a player inside the game.
* **GamerService:** Manages the creation and retrieval of games, teams, and players.

The classes related to each other:

* A **Game** contains multiple **Teams**
* A **Team** contains multiple **Players**

The object-oriented programming principles demonstrated within the diagram include the following:

* **Inheritance**: The **Entity** class provides a common structure for **Game, Team, and Player** classes.
* **Encapsulation**: Attributes of each class are private and accessed via public methods.
* **Polymorphism**: Allows the use of common methods from **Entity** class in derived classes.

These principles help fulfill the requirements efficiently by promoting the reuse of code, minimizing redundancy, and overall enhancing the maintainability.

**"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** | MacOS servers are less common but host web based applications. Mac servers are often more expensive due to hardware cost. The licensing cost can be more expensive than linux but are comparable to Windows | Linux is a popular choice for web servers due to its stability, performance, and cost efficiency. It is widely used in enterprise environments. | Windows servers are widely used for web applications. They offer robust support and a familiar environment for many developers. However it often comes with a substantial licensing fee. | Mobile devices are not typically used to host web applications but rather to access them. This makes their role in server-site hosting minimal. |
| **Client Side** | Developing for MacOS requires specific knowledge of Apple’s development ecosystems, including Swift and Xcode. Cost can be higher due to hardware requirements. | Linux client support requires ensuring compatibility with various distributions. Development might involve extra testing and adjustments. | Developing for Windows is fairly straight forward due to the extensive documents and tools. Compatibility testing across various versions may be needed though. | Developing for mobile platforms like Android and IOS requires expertise in mobile development frameworks. Cost can include development tools and device testing. |
| **Development Tools** | MacOS development is centered around Xcode, Swift, and Objective-C. These tools are well integrated but require Mac specific hardware. | Linux development can use a wide variety of tools (CCC, Clang, Python) Many are open source and free, which reduces the associated cost. | Windows development is supported by tools like Visual Studio and .Net. Licensing for some tools can be costly, but the ecosystem is well documented. | Mobile development for IOS requires Xcode and Swift while Android uses Android Studio with JAVA/Kotlin. Cross Platform tools, React Native & Flutter, can unify development efforts. |

## 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**: I recommend using Linux for the server-side due to its cost-effectiveness, stability, and performance. For the client-side, supporting all major platforms (macOS, Linux, Windows, iOS, and Android) ensures broad accessibility. Cross-platform development tools can help manage this complexity.
2. **Operating Systems Architectures**: The Linux server architecture will use a standard LAMP stack (Linux, Apache, MySQL, PHP/Python). For clients, modern HTML5/CSS3 and JavaScript frameworks (e.g., React) will ensure compatibility across all platforms. Mobile apps can be developed using cross-platform frameworks like Flutter or React Native.
3. **Storage Management**: A cloud-based solution like AWS S3 or Google Cloud Storage can be used to ensure scalability and reliability. Using a relational database like MySQL or PostgreSQL will handle game data efficiently.
4. **Memory Management**: Linux provides robust memory management techniques, including virtual memory and efficient allocation strategies. This ensures the Draw It or Lose It software runs smoothly even under high loads. Client-side applications will rely on the browser's or OS's native memory management capabilities.
5. **Distributed Systems and Networks**: Implementation of RESTful API for communication between different platforms. The use of WebSockets for real time updates and notifications. Make sure redundancy and failover mechanisms are in place to handle connectivity issues.
6. **Security**: Security measures will include HTTPS for encrypted communication, OAuth for secure authentication, and regular security audits. Using platform-specific security features (Keychain for iOS, Credential Locker for Windows) will protect user information. Regular updates and patches will be essential to maintain security across all platforms.