

Draw it or Lose it!

# **CS 230 Project Software Design Template**

Version 3.0

## 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 | 09/02/24 | Nathaniel White | Module 3 – Prototype |
| 2.0 | 10/05/24 | Nathaniel White | Module 5 – Operating Systems |
| 3.0 | 10/15/24 | Nathaniel White | Module 7 – Recommendations |

**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)

Currently, the game is an Android-only application. In order to facilitate a multiplayer game in a web-based distributed environment, consider a client-server model to facilitate socket connections with a well defined protocol. In order to make this transition, many components to the application must be changed such as it’s architecture, memory management, and security.

## Requirements

* Many teams can play a game.
* Many players can be on a team.
* Teams and Games must have unique identifiers/names.
* There cannot be multiple instances of a game at the same time.

## [Design Constraints](#_2et92p0)

* **Client-server:** A client-server model should be implemented to create socket connections between clients in different areas. The data cut off between the client and server depends on how the game is meant to be played. Meaning, does a client have a single player, a whole team, or a whole game? This cut off point will affect the performance of the server and scalability.
  + For browser compatibility, I recommend javascript/typescript for the client since there is not much computation to be done on the client side.
  + Java, Python, and Go are all great languages for the server backend and are well known by many developers.
* **Stock Photo Database:** Rather than searching through an unsorted library, a database should be implemented to speed up image fetching. This can be done using software such as MySQL, SQLite, or Postgres. Combining this with lazy loading (waiting to load/fetch an image until it’s needed) will rapidly decrease wait times for game startup.
* **Application Architecture:** The application is going to move from operating purely on android systems to any system that can operate a browser. The application logic can be moved to a server using either owned hardware, or cloud-based hardware. Then, an Application Programming Interface (API) can be implemented using Representational State Transfer (REST) protocols such as HTTP, HTTPS, TCP, etc., in order to expose it to clients seeking a connection.
* **Storage and Memory Management:** Because a large library of photos will be used, the server will either need more storage or it will need to connect to a database where the photos are held. Regardless, storage specifically for these photos will be necessary. Moving the photos to a database protects the development of the game from the implementation details of how the photos are stored such as indexing or partitioning. However, if the library is small enough, it may be preferred to have the photos stored on the server to improve image fetching speeds (no database connection means less network traffic).
* **Security:** The application’s security must be improved to ensure all clients connecting to the server are ‘good’ clients. This can be done by requiring tasks performed on the server to verify the clients credentials before performing such tasks.

## [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)

* **Inheritance:** Games, teams, and players are entities. They hold the id and name member variables and accompanying methods just like Entity does. This makes it simple to verify that all names, and therefore instances, of an object are unique.
* **Composition:** A team can have many players. A game can have many teams. A game service can have many games. These relationships will make developing unique identifiers much easier since you call the method of one object to create an object that is of the same type as the applicable member variable.
  + i.e., gameService.addGame, game.addTeam, team.addPlayer.
* **Encapsulation:** Since none of the classes even have mutator methods (other than object instantiation), the data is very well encapsulated. This will ensure consistent values for data. Pairing this with the fact that every entity must be unique as per the design requirements, there should only be ‘good’ values in memory.

"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** | While it *can* be used for servers, this use for Mac would be atypical. Like Linux distributions, it is also Unix-based, making it suitable for hosting web applications.  Advantages: as a Unix-based system, it is secure and stable.  Weaknesses: There are very few options for hardware to run MacOS on, this may make it more expensive than Linux-based options. | The most popular choice for servers, whether that’s web hosting servers, media servers, or even home automation servers.  Advantages: low-cost, diverse utility, and community support.  Weaknesses: May require more technical knowledge to setup and manage. | It is common to host web applications using Windows Server. Not used nearly as commonly as Linux-based solutions, but more common than Mac.  Advantages: Excellent support for .NET applications. Nearly no maintenance needed for the operating system.  Weaknesses: Licenses to use Windows legally can get expensive fast. | Because servers are computation heavy, content is usually served to mobile devices, not the other way around.  Advantages: there are many sensors on smartphones that may be useful in specific cases.  Weaknesses: Cellular connections vary drastically compared to hardwired network connections. Not to mention the server depending on battery life. |
| **Client Side** | Developing a MacOS native client usually entails acquiring Apple development tools like Xcode which require a small cost. | Developing a Linux native client can depend on the distribution. As such, knowledge of targeted distributions may be required. However, costs are typically low. | Developing a Windows native client usually requires Visual Studio, increasing costs further. Knowledge of .NET tooling may be required. | Developing an app for iOS or Android usually entails knowledge of platform specific tooling such as Swift or Kotlin. Cross-platform tools like Flutter may speed up development. Costs vary with the platform targeted. |
| **Development Tools** | The Xcode IDE is typically used for development on Apple products. It supports Swift, Objective-C, and C++. | With wide community support comes a wide range of tools and languages. Popular IDEs are IntelliJ IDEA and Eclipse. Some text editors can be configured to behave similar to IDEs like Neovim, Vim, LazVim, and Visual Studio Code. | Visual Studio is the primary IDE for Windows since it was made by Windows themselves. It supports the C-family of languages and more. | Tooling for iOS is similar to MacOS. However, knowledge of Swift may be useful. Android however uses Android Studio in combination with Kotlin and/or Java. |

## 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 the Java platform to build cross-platform clients which would interface with a Draw It or Lose It server. For these server(s), I recommend using AWS (Amazon Web Services) in order to host a Linux distribution where server requests will be handled.
2. **Operating Systems Architectures**: The JVM (Java Virtual machine) can run Java code regardless of operating system, as long as there is a JRE (Java Runtime Environment) installed. The JVM also enforces memory safety, security, and other items needed for logic to behave as expected. Using AWS for the server(s) will ensure scalability and remove overhead regarding system design.
3. **Storage Management**: The Java platform can be used to access file systems as well as databases. Since the server will exist in the cloud, storage may be on either the server or a cloud hosted database, depending on the space necessary to hold the image library.
4. **Memory Management**: The JVM uses garbage collection by removing references that aren’t needed in respect to the scope rules in Java’s syntax. In other words, it enforces memory safety by making sure that memory cannot leak. The JVM also automates memory allocation. Automatic memory safety and allocation allows developers to remain focused on application behavior. It should be noted that garbage collection affects performance. That said, it is rarely an issue save for extremely computationally heavy programs.
5. **Distributed Systems and Networks**: Using the client-server model, the Draw It or Lose It server(s) must allow port connections with authorized clients in order to handle server requests. By putting the server(s) in the cloud, The Gaming Room management has some say as to where they want server(s) to be geographically. This will allow management to collect data on where the majority of their users are so that they can scale servers in different regions accordingly. Data being sent over the wire should be secured using end-to-end encryption. This can be achieved using TCP/IP or HTTP standards with tools within the Java platform. The servers will also require maintenance to guarantee connectivity and respond to outages and bugs.
6. **Security**: The Java platform has many tools to ensure safe security practices. Whether it’s encryption, authentication, or authorization, Java can be used to implement it without starting from scratch. Further, Java doesn’t require any additional configuration for these functionalities such as external libraries – it’s by default. In general, data being sent through a port connection should be encrypted, and users should be authenticated and require authorization for server requests.