

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/11/2022 | Nick Mancari | Initial Version |

**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 team wants to build a web-based multiplayer game that utilizes a timing scheme as well as multiple teams. Making the application aware of multiple players, teams and games at any given time will be wholly necessary in the applications construction.

## [Design Constraints](#_2et92p0)

The biggest design constraint would be the infrastructure cost and approach, but this constraint will be affected by the technical decisions that get applied to the construction of the product. Another constraint for the software design would be memory management with keeping track of the objects in storage.

## [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)

**"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.**

The domain model outlined above in the UML Diagram highlights the classes and objects that will be used in the design of the initial software structure of the game. There are a few components that are interrelated to each other so that the objects can work in their constructed software purpose. The biggest class being the *GameService* class with the *setters* and *getters* to access and change *Game* objects. Then each object of *Game*, *Team*, and *Player* are contained under the Super Class of *Entity*. The *Entity* object then will utilize unique ids to track and maintain the related class objects.

## [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** | The hardware is expensive and hard to maintain, requiring specific hardware to run on the OS. The OS is Unix-based, making it easier to use with development. | Cheap and powerful. Lots of free and open-source tools. Can be cumbersome to maintain and support depending on the type of distribution. | Support contracts are expensive, but servers aren’t used as widely. Hardware can be on the cheaper side. | Not practical. Limited hardware resources for server software. |
| **Client Side** | Limited customer coverage. MacOS doesn’t typically target game users. | Very limited customer coverage with varying types of hardware running on the OS. | Very wide customer coverage. Computers are cheap with small limited hardware requirements for the operating system. | Widest customer coverage, most average person has one. Requires the frontend development to be constructed in two very different languages. |
| **Development Tools** | All major IDEs are supported with MacOS. Additionally, MacOS has access to the Unix tool chain and development environments. | Not every IDE has a Linux variation. It has access to the Unix tool chain and development environments. Support for the OS can be few and far between. | All major IDEs are supported in Windows. VScode, the leading IDE, is designed by Microsoft. The Unix tool chain is not available on Windows. | You could utilize applications that could emulate a development environment, but it wouldn’t be practical. |

## 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**:
2. On the server-side, I would recommend building the backend with a Linux distribution. This gives ample flexibility and optimal development environment, making software deployments easier. On the client-side, using a web browser as the client interface will simplify the frontend aspects of the development and give the user the easiest experience with utilizing the game.
3. **Operating Systems Architectures**:
4. When comparing what operating system works best with the construction of our application, there are a number of factors to consider. For example, Linux is a great operating system for developing because it supports development tools seamlessly, such as version control software like git and remote network tools like ssh. Aside from the tools available at the application level, the architectual components available for the Linux operating system are ideal for web server support. One of them being multi-threaded processors, handling multiple tasks at a given time.
5. **Storage Management**:
6. Ideally, the use of a traditional storage framework would work fine here. If storage becomes an issue over time, migrating the storage to a cloud-based warehouse might be a feasible option given the flexibility it affords.
7. **Memory Management**:
8. Linux has a wide range of benefits for optimizing memory, especially for a web server. Specific to the web application, the holding of *GameService* information in application memory can grow depending on the number of participants. Each process that handles the application information will have to be aware of each other to avoid overloading or corrupting the game data. Linux, on the OS-side, gives further flexibility on expanding memory when needed. Aside from adding physical memory, utilizing virtual memory through storage space with tools such as SWAP can give added benefits when needed.
9. **Distributed Systems and Networks**:
10. A distributed system can be cumbersome to maintain when components are broken apart and autonomous from each other but allow for modularity and futureproofing. Abstracting certain facets of the application to allow for encapsulated pieces will guarantee the mutability of changes whether they are hardware or software based. Additionally, interconnecting layers via network require the application to handle port components and packet exchanges. This is not something that typically has to be considered when building an installable monolithic application.
11. **Security**:
12. Security is something that should be considered with every piece of software no matter how complex or simple. Things to be considered should be how user information is stored in storage. User info should be encrypted to guard against malicious tampering and retrieval. At the network level, the use of TLS certificates on the client end needs to be deployed to secure the information shared between the client and the server back end.