

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 | 09/18/21 | Blaze Halderman | Executive summary (design problems/constraints), Design Constraints, Domain Model (UML diagram explanation) |

**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 software design requirements are as follows. 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.

Having a game with one or more teams involved means creating a structure for each team and having a finite number of players on each of those teams. Some conditions to think about are how many teams can play at one time? Are there specific constraints to these teams? Are these teams all individual and competing against one another? Will there be a leaderboard system which keeps track of teams scores on a global leaderboard? Will these teams be locally played, or will there be a client-server interaction aspect to the game (online)?

Knowing a team will have multiple players on it, we will want to define what a player is and how they interact with the game. The players are associated with a game entity and a team. The players also have a specific unique identifier which helps locating a specific player. Players will also have a name.

In order to make Game and team names unique we must inherit the singleton pattern which only allow for a single instance of the class to exist at any given point. This will protect the ability to have a unique name for games and teams. There will also need to be a database for storing all current game sessions in creation and team names associated which will need to be searched efficiently in order to verify name usage for game and team names.

Games, teams, and players all need to be uniquely identified and exist in memory, which would mean that a game associated with a team and its players are unique to the game itself, and the team and its players are unique to each other.

## [Design Constraints](#_2et92p0)

Design constraints must consider the scale of their game in terms of how many games, teams, and players are going to be playing at any given point in time. This can give some sense of how much memory will be required and will most likely require the use of virtual memory to provide dynamic performance based on player counts and active game fluctuations. Will require the use of a client-server relationship, with the use of a database, an API for which all the clients can request their data for games. A game must also be in sync with other teams and their players, which would require a game to only be played with the exact number of players who initially created it in order to avoid memory corruption and differences in game states. This will avoid data corruption and desyncing between different game state versions.

Another constraint to consider is whether the web-based application will create a host on a centralized server owned by the company or whether it will be hosted dynamically through a local machine. This can avoid spending excessive costs on server upkeep and maintenance, with the downside of potentially causing some players to have a lack of quality since the connection or game will be dependent on the connection quality of the host. This also will lead to issues relating to where the data will be saved and how it will be saved, leaving the idea for user creation as more static and general use versus a more personalized experience with user accounts.

## [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 program begins with a parent class defined as Entity, which privately stores an id and a name. This Entity class acts as a prototype for all our game definitions (a Game, a Team, a Player). The Entity class contains a default constructor and a constructor for initializing an entity with an id and a name. There are also getters and setters for retrieving Entity class specifics such as an id, name. There is a single method on the Entity class for converting to a string, which presumably would be used for the id and the name variables.

Player class inherits from the Entity class with the addition of a constructor including an id and a name for a player, which also has a string conversion method. The Team class inherits from the Entity class but has a privately accessed list of Players. Each Team has a constructor with an id and name, and the methods for adding players, along with a string conversion method. The Team class has a 0 to many relationships with players, meaning that there could not be a team or there could be a team with 0 or more players.

Game class inherits from the Entity class with the addition of a constructor including an id and a name for a game. There is a privately accessed list of teams stored on a game, along with a way to add teams to the list and a string conversion method. There is a none to many relationships with teams between a single game and teams, meaning there could be no teams or many teams on a single game.

Each game service object created is directly related to a game with a service not existing or existing with many other services on a single game. The game service class stores privately accessed variables such as a list of games, a next game id, next player id, next team id, and a service itself (singleton, one service at a time). The methods stored on a game service include a default constructor which is privately accessed allowing for the singleton design concept, and publicly accessed methods such as getting a current game instance (active game service), add a game, various getting game methods (based on id, name or count of games), and getting the next player id and team id.

The program driver class houses the main method for using a singleton game service, which runs and uses the singleton tester class for running the test singleton method. The test singleton method runs the game instance and allows for the beginning of a game.

**"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 must 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** | Simple UI, security at the expense of limited customization for hosting a server, limited scalability based on limited resources for expanding servers’ capacity, simple to implement since Mac is built off a version of Linux, AWS server hosting can be great for deployments (simple, powerful and could base), Apache web servers, encryption, SSL and TLS protocols, 128-bit strong cryptography, flexible security controls, | Has some similarities to Mac but with more control and less features for handling automatic or intelligent memory management, security at the expense of very minimal pre-built features, meaning would need to be built from the ground up, would allow for a more intelligent system and better memory management in which the OS handles a minimal amount | Easy to use and fast implementation at the expense of security since there is the ability to customize and create your own software on the operating system, expandable memory and scalable size amounts with the ability to add additional amounts of memory at any time, azure is a cloud-based windows server much like AWS, powerful encryption (2048-bit keys), VM, can host both windows and Linux, VPN, load balancers, PostgreSQL databases. | Limited customization and limited security, operating power is primarily for the function of the OS of the mobile device, no option to expand or scale servers, easy to use but hard to implement since each mobile device is different and requires different features or connection specifics, low cost to generate own web server (due to size), expensive to implement, limited options for dedicated mobile device server hosting, basically custom built is the way to go |
| **Client Side** | Cost is low and implementation could take minimal time, but also requires an expertise in the operating system and how it handles client-side software. Could require different system architecture which utilizes memory virtualization to enhance user experience, would allow for a consistent experience and stable client, will require proper packages and libraries for running the proper software and will need proper libraries and packages for running client-side software | Cost is high, implementation could take extended time to implement client-side features since system is built from lower-level machine language and difficult to find any pre-built features or packages for handling client development, will require proper packages and libraries for running the proper software and will need proper libraries and packages for running client-side software | Windows cost is low, and implementation could take minimal time since there are OS development packages and application client deployment software which can be installed. Languages such as Java and Python work well with development on Windows systems which could make implementation fast and simple, will require proper packages and libraries for running the proper software and will need proper libraries and packages for running client-side software | Mobile devices implementation for client side could cost a lot, time could cost a lot as well since there are various types of mobile devices which run off of different OS’s, expertise in mobile devices could require various levels and extensive work with different devices, could be best implemented using a web application, would require extensive work to implement across multiple platforms running different OS, will require proper packages and libraries for running the proper software and will need proper libraries and packages for running client-side software |
| **Development Tools** | Mac has various programming languages which can be developed for including Assembly, C, C++, Swift, and Objective-C. Primarily the IDEs which can be used are VIM, Atom, Visual Studio, Eclipse. XCode developer resources for building apps, Homebrew which is a package manager, Sublime text editor, FlexiHub shares iOS devices over networks allowing developers to access devices easily, iTerm2 is an open-source terminal which allows for customization and os tools, netbeans ide for easy development of GUI and debugging, Tower is a git client for managing version control, licensing can be expensive for these tools, but most are free and open source, single development teams will be needed | Linux has various freedoms when it comes to development tools, written using C, C++, Java, Python, JavaScript and Shell, PHP, NodeJS, does not support .NET related languages such as ASP. Net, C#. net and VB. IDEs are visual studio code, sublime text, sea monkey for building easy to use websites, Quanta for developing web software, BlueGriffon web editor build around Mozilla Firefox and a rendering engine, Gedit which is the official Gnome desktop editor, Vim text editor or programmers’ editor for development in shell, and netbeans, licensing can be expensive for these tools, but most are free and open source, multiple development teams will be needed | Windows has various programming languages which includes JavaScript (Full stack development), Python, Android studio for developing mobile software, windows desktop apps, C, C++, C#, docker for remote development using a variety of azure services, PowerShell which is a task automatic configuration management shell with its own scripting language. IDE’s include visual studio code, azure, .net. You can also run mac virtually on windows using a VM which is stored on the same hard drive, licensing can be expensive for these tools but most are free and open source, single development teams will be needed | Programming languages for developing mobile devices include Java, JavaScript, Swift, Kotlin, the best development tools are React Native developed by Facebook for open-source mobile app development for both Android and iOS, Xamarin is a Microsoft open-source framework that is new and used for a cross-platform mobile app development uses C#, Flutter mobile app development tool is an open-source framework which supports multiple operating systems iOS, Android, licensing can be expensive for these tools but most are free and open source, multiple development teams will be needed |

## 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**:

Linux operating system would be the best platform given the scope of the project thus far. Using a client-server architecture until the player base of the game expands, then can make the decision to integrate game platform onto a more significant server such as AWS or Microsoft Azure. This will save cost and allow the focus of the game to be on the experience while focusing on costs and security. This will allow the game to be built from the ground up with more control and memory management capabilities. Building the game “engine” or “platform/client” using Linux will make much better use of the long-term capabilities of the application, in which could make much more significant changes in the long term.

1. **Operating Systems Architectures**:

A client-server MVC architecture should be integrated to allow for scaling the application right off the bat and will remove system dependencies or downloads for specific aspects of the application. If the application evolves or generates greater interest, overhauls to the UI or graphics could be made, in which would make an easy integration from a purely web-based application to a client-side application running on the individuals machine. This allows for flexibility when designing the architecture and makes compatibility fluid across all platforms.

1. **Storage Management**:

Storage management will be handled using a database such as NoSQL. This server will be hosted through a private HTTP protocol web-hosting company to reduce costs upfront. Once the game gains some traction and begins to expand its player base, we can look into having more ‘physical’ locations for advanced server hosting options or alternatives.

1. **Memory Management**:

Memory management will be performed via the client on the individuals’ machine which requires the pre-loading of the resources for the site. This will be stored in the cache and allow the site to viewable in offline mode and will update once the site has been connected to the server. This will allow the creation of APIs to pull data for specific features of the application, making memory management efficient and only used when necessary. The use of APIs also contributes to the long-term scale of the project and can allow flexibility during development. All relevant data for the game can be stored via the web browsers caching capabilities, making the application perform efficiently and effectively. This also will allow the application’s core front-end features to be stored on the web browser (client-side) and allow for the individuals to always access the visual features. This removes the need to consistently load the resources each time the application is loaded on the system (from server to client) and will allow the application to perform more efficiently. This keeps focus on the core game and removes the constant need for a server at-all-times.

1. **Distributed Systems and Networks**:

The game will be distributed via an individual’s internet connection and the use of a web browser through a web-based server-client application architecture. This will remove the need to customize the application to make it more compatible with individual system requirement types. The use of a web browser also allows for easy and universal storage and memory management which is handled mainly on the server-side of the application with less strain on the client-side. Connectivity will rely solely on the strength of the user’s internet connection with their local ISP (Internet Service Provider) and will also require the use of a network adapter via WiFi or Hardwired via LAN. Hardwired is the preferred method since it reduces the risk of disconnection and will provide the user with the better-quality experience. The user will also depend on the servers to run consistently with no down time, always making the data for the game and the game itself accessible. This also allows for continuous updating of the application without the need for downtime. Handling maintenance on the application can also be simple since the application maintenance can be minimal.

1. **Security**:

Security of the application will be secured via a username and a password for each individual user along with another form of verification. This other form of verification will be the use of a seed or 2-factor authentication. This will provide the most secure environment and experience for the users. The user’s data will be stored in secure database encrypted using 256-bit encryption. Each user’s game will be saved via server storage, but each session will be randomly generated to avoid hijacking or middleman access via brute forcing the URL address. There will also be user tier list and access rights for specific contents of the application. A user will be required to login and validate each session (each time the web browser is opened, will be saved with the current web-browser session). Admin access will also limit the amount of control on users to avoid hijacking or admin-control-rights abuse. This will provide the most security to the users and a balanced amount of power for the admins. An admin will not be able to update a user’s credentials immediately and will require the user of a “special authentication method” using 2-factor authentication to first request the changes and then requires another admin user to confirm the access, in which also will require 2-factor authentication from the other admin. This will ensure that the proper access is given to users and that a user intending to gain access is indeed an admin with the correct access rights.