

# **The Game Room**

# **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 | 07/17/2022 | Joe Thompson | Initial Update |

**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 application is currently only available on Android. The client wants us to design a similar game that is web-based. This is because, in being web-based it would be available on multiple platforms so that everyone can play it. The current team needs help developing processes and environments to develop the game within. As such we will need to implement some DevOps best practices to get the ball rolling.

1. The application must be available as a web-based game.
2. For any given game, there must be the option to have one or more teams.
3. For each team there must be more than one player assigned to it.
4. No two teams or players can have the same name.
5. There cannot be more than one instance of the game in memory at once.
6. Each game, team and player must have a unique identifier.
7. New processes and environments must be implemented to aid development.

## [Design Constraints](#_2et92p0)

1. A large amount of web servers will be required to host and server the web based game to web traffic.
2. There needs to be an ample amount of memory to service the teams and players that are instantiated into memory.
3. The application will require a large amount of storage to store game data.

Depending on the findings of an as-is assessment, there may be a good amount of the codebase that we can reuse for development of the game into a web-based application. However, we cannot reuse the code if it was developed in an android specific framework. The networking will take additional consideration as well, as a distributed web-based environment is different than an Android deployment.

## [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)

For starters the Entity class is the parent class which holds all of the common fields shared by the Player, Game, and Team classes. This is an example of Inheritance. Particularly, the id and name fields as well as the base constructor, accessor methods, and the toString method. This approach helps to prevent overcrowding of classes or repeat fields in related classes. This could become a problem if you decide to take a different architectural approach to your program design, change the field in one of the classes, but forget to change them in the other. By using inheritance, you only need to change the field in one place.

There are aggregation associations between GameService and Game, Game and Team, and Team and Player. Specifically, a GameService can have 0 to many Games, a Game can have 0 to Many Teams, and a Team can have 0 to many Players. In the UML diagram this is represented as an association with a plain line, however, I feel this is really an aggregation relationship, because a Player technically could exist without a Team and a Team could technically exist without a Game.

The toString method is being overridden by the child classes of Entity. In my implementation, we are still able to access the private fields of the parent class via the public parent toString method which we are able to reference in the child toString method via the “super” keyword.

Encapsulation is used here to hide certain functionality from being accessed by other classes. In particular, the default constructor in the Entity class is made private, so that others cannot instantiate an object using this method. Furthermore, many fields throughout the classes in this program are also made private for the same reason. Furthermore, in the GameService class, the constructor is made private so that the class itself is the only one that can instantiate an instance of itself. We use this in the GameService class because this class is utilizing a Singleton design pattern and we only ever want one instance of this class to exist in our program.

In the Singleton GameService class, the UML diagram utilizes a variety of static fields and methods. When a method or field is static that means it belongs to the class itself, not an instance of the class as is the case with an object. The GameService class then has a static “getInstance” method to access the one instance of itself which is capable of creating and retrieving games it creates. The GameService class also manages the counters used as unique id’s for Players and Teams. These unique id’s can be accessed by other classes via the public getNextTeamId and getNextPlayerId methods.

The ProgramDriver is disassociated from the other classes in the UML Diagram, as it is simply the driver. However, it does have a “uses” relationship with the SingletonTester which it calls to test the functionality of the rest of the classes. It is worth pointing out that the Driver has a Static method itself, main. This is due to the fact that the progam needs some entry point, and a static method doesn’t require an instance of “ProgramDriver” to be created. If this method, was not static, then what class would run to create an instance of “ProgramDriver” to then run the rest of the program? It wouldn’t work.

The final thing I want to point out about this UML diagram is the way in which methods which add an item to an iterable, also return the item they add. This allows us to then quickly access the item we just added as was the case when creating a game, and then accessing the game to add several teams. This is a pseudo-design pattern (not one of the main design patterns) that I will personally remember.

**"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** | There is a lot of extraneous software on a mac operating system unneeded by a server that will take up considerable storage, memory, CPU, and other resources that could otherwise be used by the server for tasks relating to the application.  There are some licence costs for the OS X Server up to $900.  Pro: Very resilient and thoroughly tested system due to large commercial use and a relatively homogenous product (as opposed to Linux with a million, less thoroughly dished out distributions). | There are Linux distributions that are as bulky (tons of software installed) or as lean (very minimal OS, with only the services you need to run your application installed) as you need.  Linux is less user friendly for server admins since it is not generally accessed through a GUI, but a terminal.  It has a large library of fantastic CLI utilities from the open-source community. It possesses a solid track record of being used as servers in the cloud. One of Linux’s greatest strengths is that it is an open-source OS and as such has no licencing costs. However, for some enterprise distributions like Red Hat, there will be some fees for use. | Like macOS, it has a lot of software installed that is not needed for a simple server. This takes up storage and compute resources. Not all software is going to be compatible with Windows.  Most importantly for us however is that there is steep licencing cost for a windows server. For the datacenter Edition, it is $125/month to lease and $6,155 to own.  Windows has a friendly UI second to none. | Mobile OS’s were not designed to be servers, but clients. A Mobile Device is not equipped with a large quantity of compute resources such as RAM, CPU, and Storage.  However, the mobile device is very portable. |
| **Client Side** | Safari offers extensions that need to be kept up to date. Depending on the extensions installed in the end user’s environment, this could cause nasty bugs for end users.  Mac has excellent integration with the Firefox and Google Chrome Web Browsers. | There is no “one” Linux distribution, and as such, your application may work well on one distribution but not another, which makes fine tuning your web app to run on “all” Linux machines, infeasible.  Linux does not have the most robust integrations with the Safari and Google Chrome web browsers due to being open-sourced.  There are multiple Linux GUI’s in use such as GNOME, KDE Plasma, LXDE, and XFCE. We need to ensure that our application runs correctly on web browsers for each of these environments for full coverage.  Linux is more commonly used without a GUI or web browser, and interacts instead with the web via commands in the terminal such as curl and wget. As such, you do not need to worry as much about your application rendering correctly in a web browser because often none is used. | Windows uses Internet Explorer, which is famous for not supporting all of the newer features that a lot of browsers do. For example, Internet Explorer does not support async functions which were added to Javascript in ECMAScript 2017. As such, IE has very specific issues that need to be accounted for in development that other browsers just won’t have. This is a huge hassle, and it will take an inordinate amount of time to get the application running correctly on IE compared to all the other browsers on other platforms.  Lastly, certain browsers are not supported on versions of Windows before version 7 such as Chrome and Firefox. | In the case of a mobile device, people generally prefer to access apps via a mobile application, not a web browser. While it is possible to open a web browser on a mobile phone, the size of the screen can yield some un-wielding results such as the components not stacking properly and the screen seeming too zoomed in. As such, the application must be developed to adjust for very small screen sizes. |
| **Development Tools** | Popular open-source containerization alternatives to docker such as Podman are not yet available on Mac. This is an issue because docker now requires that you pay if you are using it commercially. Mac’s do not have support for Visual Studio Code (Only VS Code). Must spend lots of money on third party software to get certain functionality. Some excellent, high-performing development software will only run on a mac. There are restrictions in the Apple app store which makes It difficult to distribute new software to other developers. | Linux has a giant library of powerful utilities available to developers for free. GUI’s in Linux are often buggy and may have difficulty running IDE’s when running an older version of Linux with, say, a newer version of Visual Studio Code. Since Linux is an open-source Operating System, development is constantly occurring and things have the potential to change fast. A given IDE or development tool may work fine one day, but stop working after an update. | Open-source containerization alternative to Docker, Podman, is available on Windows. Windows lacks the large library of powerful utilities that Linux has. Windows uses Powershell which is a shell unique to Windows and many developers are not as familiar with Powershell as compared to the Linux CLI. It may take time for them to learn. | You can use a wide range of languages and frameworks to develop a web app that will be accessible by mac. For the frontend you generally will use Javascript or typescript and a frontend framework like React or Vue. For the backend there are as many languages and frameworks to choose from as there are grains of sand on the beach: Springboot, Django, Flask, Express, etc. | |

## 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 will be the ideal operating platform to serve the web application from. It is a tried and tested solution for serving web applications. Furthermore, it comes in small sized distributions that will allow us to deploy the application with as little extra software installed as possible. This frees up resources such as memory and storage on the machine and removes complexity which could cause things like configuration drift. Lastly, there is ample support for Linux instances on all cloud platforms.
2. **Operating Systems Architectures**: Linux is an Open-Source Operating Systems which will allow us to run our application on it with no licensing fees, with the exception of if we use a Red-Hat Distribution. This also entails the support and upkeep by the operating system with a large community of talented developers. Linux utilizes a monolithic kernel, meaning it doesn’t incorporate layering or submodules extensively. This can make it difficult to debug, yet very efficient and well performing. Linux is a multi-programming and time-sharing system which has users and a superuser. Linux systems use kernel threads and in particular the Linux uses the Pthreads implementation of the POSIX standard. We need a fleet of devoted servers to serve and run this application.
3. **Storage Management**: I recommend using a distributed database service in the cloud to ensure that writes are ACID compliant and also to ensure that our data is highly available and resilient against the failure of any one node. Storage services like this are also easily scalable, due to the high amount of abstraction taking place in virtualized environments like those that exist in the cloud. To supplement this, hold some data locally and utilize a NAS drive to manage the movement of certain mission critical files between our on-premise and cloud servers. To ensure redundancy on any local hard drives we incorporate into our on-prem storage we could utilize a RAID array which ensure the recoverability of data if any one hard drive fails at a given time.
4. **Memory Management**: It will be key for the application to manage memory efficiently by releasing memory it is no longer using. A good example of this is the application quickly releasing an image from a previous round from memory after that round is over. That way, there will not be a build-up of images not being used In memory. Long-term storage of these images is where they should usually reside if not currently being used. It takes time to move data from storage to memory. To compensate for this, some form of caching should be used to load more than only the current image into memory at a time. This is because, we don’t want to wait to start loading an image from storage once an image has been guessed in a given round, because this could cause latency if the disk read is slow.
5. **Distributed Systems and Networks**: Cloud infrastructure in conjunction with the Linux operating system can be used to create a system architecture that is highly available, distributed, and fast. By using a distributed system, we can make sure we have servers spread across a wide geographic area to reduce latency and improve user experience. We can also utilize open-source software available and compatible with Linux to cache requests near edge locations to decrease latency. Each node in our distributed system will be able to communicate with other nodes via http requests to our server. The server will listen on a given port and the client application will communicate with this pre-defined port on ephemeral ports. If a given server node crashes, we should have health checks to detect this and some other node should automatically have the clients traffic redirected to it via a load balancer so that that instance of the game is not interrupted.
6. **Security**: When deploying our instances in the cloud we will inject any secrets or sensitive values into the application at runtime. The application source code will only reference environment variables. In our cloud storage, we will encrypt everything on write. This will ensure that even if anyone manages to get access to sensitive customer data and steal it, the data will be encrypted and thus useless. Furthermore, our application will use https with an SSL certificate to so that all communication is encrypted at layer five of the OSI model. It is a multiuser system, which facilitates managing security. Each file or directory that is made by a given user is given a default set of permissions for the owner of the file itself, the members of the group to which the file belongs and all other users. This setup is ideal, because even if a third-party were to get access to the user, any given user only has access to a certain subset of files, not every file in the system.