

CS 230 Project One Game App

# **CS 230 Project Software Design Document**

Version 1.2

Parker Harb

## Table of Contents

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

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

[**Document Revision History**](#_3znysh7)2

[**Executive Summary**](#_2et92p0)3

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

[**System Architecture View**](#_3dy6vkm)3

[**Domain Model**](#_1t3h5sf)3

[**Evaluation**](#_2s8eyo1)3

[**Recommendations**](#_3rdcrjn)5

## [Document Revision History](#_3znysh7)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <03/16/22> | <Parker Harb> | <Added required features for game to be playable by multiple teams with one instance of game at time, added basic info to doc> |
| 1.1 | <03/28/22> | <Parker Harb> | <Added considerations for server and client side needs, as well as possible challenges> |
| 1.2 | <4/14/22> | <Parker Harb> | <Refined recommendations and removed instructions> |

## [Executive Summary](#_2et92p0)

<The development for the product, the web delivered game, will be documented here. It’ll be in java for now. There will be use of the singleton design pattern for the game class. Further information will be written here as development moves onward. The backend of the system will be in java primarily. The front end will be built with HTML5, CSS3, and javascript.>

## [Design Constraints](#_tyjcwt)

<For web based environments you have to ensure the game will function well on a variety of hardwares, and browser softwares. The design must be effective as well as consistent across environments, thus careful planning is required. It’s important to implement a scalable system that is efficient across large numbers of players.>

## [System Architecture View](#_3dy6vkm)

## [Domain Model](#_1t3h5sf)

So all the objects obviously inherit from the original object class, java.lang.object , as always in java. With the actual relationships beings from Object -> Entity -> Game/Team/Player. There’s a relationship between the game and gameservice classes, specifically the creation of game instances requires the use of the game service class. I actually also built into the game service class the ability to create teams, so for my code there is also a relationship there. All the instances are also linkable with each other, games containing a list of teams, teams containing a list of players.

**"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](#_2s8eyo1)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | I don’t recommend using a mac for a server, but if you do, you will have the advantage of having access to apples robust tech stack, which is an advantage for any project. If I were to rank different systems in viability for a server, I’d put Mac as #2. | Linux dists for servers are generally light weight and most people know how to use them, so I recommend using linux for a server. Easy number one spot. | For windows, you’ll similar to apple be using the microsoft tech stack to develop, that being said, I prefer apple tech to .net, though that might just be personal preference. I find that apple is far more modern in it’s approach. | Do not host a server using your phone. |
| **Client Side** | Mac is an apple product and thus to deliver a product to it you need an apple machine. There are obviously testing challenges to this as well. That being said, this isn’t as big a problem if you’re delivering via the web. What will be a big issue is apple’s far more specific security measures. There needs to be thorough testing using apple products to ensure the average user does not encounter serious challenges. | Delivering to linux is difficult. Probably not all dists are going to run your app, but you can cover the major ones. Some orgs opt not to support linux at all due to the relatively smaller userbase. For a web app, I’d say it’s very easy to support linux though, as it becomes less a question of operating system and more about what web browser is used. | Supporting windows is relatively easy, though there are testing challenges involved. I recommend testing the web app on multiple popular browsers, as well as creating working demo version at least once a month to ensure quality. | Supporting mobile clients is a challenge that should be faced from day one, with responsive design and touch based controls. Ensuring that the same web app functions for both a phone browser like safari as well as a web browser like chrome is an extremely difficult/sometimes impossible task. Still, having lots of testers and experienced developers helps. |
| **Development Tools** | You’re gonna be very familiar with xCode if you develop with mac. That being said, you probably don’t need multiple teams for the different operating systems. You may need different teams for the front and back end, but even that is still going to be dependent on the skills of the team members. | I recommend intellij if you want a fully featured IDE and visual studio code if you want a text editor with quality of life features. I generally use visual studio code myself. As for testing, for the frontend I recommend using googles testing systems such as google lighthouse. | Same recommendation as linux honestly. On the subject of licensing costs, I generally recommend using free and open source software wherever possible. I don’t like spending money. That being said, in order to host your game, you’ll need some kind of cloud solution, aws or microsoft azure or something along those lines, they cost money. | You need to download a virtual device for all the various phones people use. You should also have physical models for testing, as in my experience the virtual devices don’t always represent all the challenges you’ll face. |

## Recommendations

1. **Operating Platform**: <For server, ideally you should use a linux dist such as ubuntu. This will be your most light weight and cost effective option, it’s also well supported and commonly known, so finding engineers to operate and work with it should be easy. You could also opt for a mac with some kind of apple operating system, which is a viable alternative. I don’t recommend windows as an os for a server, as it is non standard.>
2. **Operating Systems Architectures**: <You’ll need a server that can handle lots of users and clients dynamically. That is why I recommend using AWS, Microsoft Azure or some other cloud based system with an infrastructure as a service model. You can pretty easily configure them to dynamically allocate server space for your system based on time of day and various other factors. They also make it very cheap to spin up a small system and scale it for whatever your needs are. You can, through either built in gui or command line, access said server as well online. This makes the whole process very streamlined.>
3. **Storage Management**: <For this, I recommend mysql db, integrated with aws s3 and aws aurora. Another option would be something like mongoDB, but since this is in the context of an online game, I believe a relational database is more appropriate for storing the various game states as well as user information. mysql is a very common solution for storage management and it is thus very easy to find engineers and scientists who are experienced with its use. Likewise, aws systems are also quite common and provide similar advantages. >
4. **Memory Management**: <Players account will need to be pulled from the database based on use. You could even move player data that hasn’t been touched for a long time to a server that specializes in long term storage. The game states will also need to be kept for a short term bases. This can create a bottleneck if not controlled for. With a cloud based solution, one can configure the system to dynamically adapt itself by predicting when users will be online. Probably most players will be playing over the weekend, during the daytime. As such, you need to account for this in your system.>
5. **Distributed Systems and Networks**: <This isn’t really as big a challenge as it used to be. You don’t have to do anything complicated, just link up a backend to a database, preferably using an object relational mapper, and link that to a cloud based architecture, then build out a web based interface and you should be good. For the actual technology recommendations, I would recommend the following.

* Client side: js,HTML5,CSS3. Possibly a js framework with typescript.
* Server side: Java with the spring framework, or a small node js or python/django app. For the scale of this project though, and considering how much business logic will be kept to the backend, I must recommend java over alternatives as it’s an industry standard. Being compiled and allowing for multithreading is just very important.
* For the cloud based architecture, I like aws, but from what I’ve heard microsoft azure is also quite good. There are some challenges associated with ensuring stability across systems, as such a robust and experienced testing team will also be required. There are companies you can outsource testing to, if you don’t want to opt for a dedicated team.

1. **Security**: <On client side security, you need to ensure that any user input is properly sanitized before you take it in. That means controlling symbol inputs both on the front end and back. You also need to ensure that endpoints have some kind of safety measure against bad actors, such as authorization keys being created for users. As for the server, security needs are going to be a focus from day one, and everything must be designed around preventing large scale data leaks. For cloud architecture, I use AWS for my server needs, so I’ll explain in that context. You need to set a root account, which is your main account. Create a long password and set up multiple verifications systems. Passwords should be rotated and should be kept by a limited number of personnel. Most users of the system should be given IAM accounts with access to whatever it is they need to access. The people with access to the root account should also generally be using IAM accounts for most of their work, the root account primarily used to create those IAM accounts. On the subject of user accounts, a multifactor system would be ideal. Require the user to retain a password of high complexity, and include a system to email and/or text one time codes to users to ensure the users identity.>