

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.1

Joey Wheeler

## Table of Contents

[CS 230 Project Software Design Template 1](#__RefHeading___Toc1771_2007523891)

[Table of Contents 2](#__RefHeading___Toc1773_2007523891)

[Document Revision History 2](#__RefHeading___Toc1775_2007523891)

[Executive Summary 3](#__RefHeading___Toc2345_489112344)

[Requirements 3](#__RefHeading___Toc2347_489112344)

[Design Constraints 3](#__RefHeading___Toc2349_489112344)

[Evaluation 4](#__RefHeading___Toc531_4253260929)

[Recommendations 5](#__RefHeading___Toc1777_2007523891)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 04/16/23 | Joey Wheeler | Initial Revision |
| 1.1 | 4/18/23 | Joey Wheeler | Compiled 3 projects into one report. |

## Executive Summary

*Client needs help setting up environment. We are to help streamline the development by providing a base environment for them to iterate on.*

## Requirements

1. *A game will have the ability to have one or more teams involved.*
2. *Each team will have multiple players assigned to it.*
3. *Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.*
4. *Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.*

## Design Constraints

*This will be created in Java, as that is what their developers have familiarity with. Should be at least a working baseline that the client can run.*

## Evaluation

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Not really relevant for server side. Mac focuses on UI/UX for consumers, better for Client Side, would not use. | Excellent Choice. Linux is known for it’s security, reliablilty, and speed. 80% of the web is hosted on Linux. Would also want to evaluate how well this can be supported within your company. Some companies are exclusively using Windows for both front-end and back-end. Would need the right talent to support. | Secure and great for Enterprise level servers. Not great for speed. Decent for reliability. Can work well at the enterprise level, however will be far too bloated for something like a game server. | Not a good choice for a server. These should be used only as a client. |
| **Client Side** | Many users will have Macs. Will need to provide support if trying to reach largest possible audience. Can decide to not support if just looking for biggest target audience. | Linux desktop environments are becoming more common. Would have to support distros that use apt and dnf/rpm for packages. Doesn’t make up bulk of consumer devices though. Would drop support for Linux before Mac. | Must support this as a client. Most consumers and businesses rely on Windows for their operating system, especially gamers. Have to support this. | Large Audience here. Would recommend supporting these users as well. If application can be made mobile friendly I highly recommend. However, if developer resources are low, I would only give up on mobile after already giving up on Mac and Linux. |
| **Development Tools** | Xcode is exclusively available for Mac, so if you are developing for Mac, it would make sense to purchase one. Unix based OS with bash terminal. Good for developers. | Distros like Fedora are specifically designed to support developers using Linux. Highly recommend. The high degree of customization allows developers the freedom to really improve their own development environment as much as possible. | Decent for development. Useable, and convienient that it will also be the same OS most clients will use. A fair choice, although not for every developer. | If a mobile client is being supported then this is essential. Both iOS and Android offer SDKs and developer tools so that you can test your application on native mobile devices and Vms. |

## 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 a web based client for your game that supports native and mobile applications. I would host all of your back-end on Linux servers. I would recommend supporting edge, chrome, and Firefox. Also support Windows, Mac, Linux, and Android. This will allow you to reach the most users possible. I would also create a version of the client that works on both xbox one x and ps5. We are really shooting for maximum user coverage since your game is really focusing on a broad casual audience.
2. **Operating Systems Architectures**: Above is the what, this is the how. Use RHEL servers for your back-end. Host them on AWS. Since you don’t really know when your game is going to take off, it is important that you can handle fluctuations in usage spikes. This is why AWS is good for your back-end. It uses elastic beanstalk for load balancing, so you will only pay for the resources you use. The back-end should utilize containers and container orchestration via kubernetes to allow you to quickly adapt to changes, push updates, and balance requests, without users suffering performance. The front-end of your application makes the most sense to be written in a performant cross-platform engine like unity or unreal engine. This requires C++ and C# respectively. This allows it to be hosted on all the platforms mentioned above.
3. **Storage Management**: Utilize S3 buckets on the back-end for storage. Use a more performant NOSQL database or JSON if you need any kind of persistence. You will need a lot of disk space spread across multiple servers to handle this. That is okay. AWS has services for this.
4. **Memory Management**: The key here is to utilize the client cache as much as possible for quick loading times. Also make high usage of AWS CDN so that all of your files that need to be served up from the back-end arrive as fast as possible. Optimization of algorithms inside of your programs as well will ensure speed. The key here is speed. Gamers demand it.
5. **Distributed Systems and Networks**: This is all handled with AWS in the cloud. There was a time where xbox and PlayStation did not play nice with each other, but that time has mostly passed. They offer services now to allow for cross-platform play. AWS is built very well to deal with outages and provide 24/7 services for monitoring. You may want to consider hiring at least 1 site reliability engineer and make sure that your back-end team receives good training on AWS. This will make or break you for your network and distributed system. It is also a good idea to handle accounts for each user that plays your game either locally or using some third party service. This simplifies a lot of the distribution when each agent has an ID.
6. **Security**: Every point of your system should follow security best practices. There are too many of these to go into here, and I would recommend hiring staff who specialize in this area. However, here are a couple of things you can do. Hire penetration testers to try to hack your system and expose vulnerabilities. Do that periodically. Use HTTPS for web-based traffic. Lock down the SSH for all your servers. Ensure all employees are trained to look for phishing attacks and utilize strong passwords containing special characters, numbers, and upper case characters.