

The Gaming Room Web App

# **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 | 11/13/2021 | Hunter Ashner | 1.0.1  Added Entity class and designed accessors to GameService class. |

**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)

This summary is intended to address the design and implementation on a web-based application to run in congruence with the previously designed Game Room mobile application. We plan to design and implement a web-based application using modern technology to give more users access to the Game Room from other devices and operating systems. Web-based applications offer greater opportunity to reach more users and provide a convenient avenue for easy access to your services. To proceed with the design process, it is important to understand that this design is exclusively for browser-based use which will alleviate the time and costs often associated with designing applications for multiple operating systems. It is also important to note that integration with your current software is of the utmost importance. Use and potentially migration of the current database will be necessary to keep data accessible from all platforms.

## [Design Constraints](#_2et92p0)

The design constraints of developing an application in a web-based distributed environment are ensuring browser compatibility as well as access via mobile devices. One important constraint is determining how the web-application will be viewed via mobile devices and if it is aligned with the model that would be seen on a native web browser.

## [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 UML class diagram below depicts the Game, Team, and Player classes all deriving or inheriting from the parent class Entity. Since Entity contains a private default constructor a call to the super constructor Entity was needed in the default constructors of the Game, Team, and Player classes. To protect private attributes of the GameService class, accessor methods were added to access and track the nextPlayerId and nextTeamId attributes. These attributes assign unique ID numbers to all new Games, Players, and Teams. Additionally, Iterators were added to the Game and Team classes to iterate through Team and Player arrays ensuring no duplicate names were being used.

**"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** | Hosting a web-based software application on a Mac server in my opinion is not the best option. To improve connection speeds, it would be most effective to host the server-side application in the cloud. Just based on access to support it would make more sense to use either Linux or Windows servers which are very reliable and much more widely available and easy to troubleshoot.  Project 2:  After reviewing my previous opinion on a MacOS server for the Game Room application I will stick by my opinion that this would not be the ideal solution for hosting the server-side application and REST service for the Game Room application. I do think I am slightly biased in my decision as I have most experience on Windows Server and Linux Servers, but after some research it seems as though MacOS server is used mainly as a domain controller among small business, and I have never in my experience seen Mac OS being used to as an app server and find that most cloud service providers do not offer Mac as an OS option when creating a cloud instance. | Linux distros Ubuntu and CentOS are the two most common operating systems for hosting web-based software applications. These operating systems are widely available and easy to install on a cloud server instance. Also, receiving support and information on potential problems would be much easier when using a very popular platform.  Project 2:  After reviewing my previous opinion on Linux based operating platforms for the Game Room server and REST service. I will stick by my opinion that Linux, particularly Ubuntu and CentOS would remain ideal solutions for hosting the server application for the Game Room. These operating systems are widely adopted and easy to troubleshoot and install, which is important when maintaining the server and prioritizing uptime. For a web-based business every second their services are down, potential profits being lost. | | Windows server is the second most common operating system for hosting web-based software applications and in most situations, it comes down to the preferences of the person managing the server or the constraints of the server hardware. Support and information for host web-based applications on Windows Server is widely available.  Project 2:  After reviewing my previous opinion on Windows Server, I will stick with my opinion and recommend Windows Server as the operating platform to use to host the Game Room’s server application and REST service. I had the opportunity just last week to use Windows IIS to host a ASP.NET REST service for a custom application my coworkers designed. It was a great learning experience, and I was able to make changes very easily to the Windows DNS settings to allow the web application to be accessed locally through an FQDN saved on their primary domain controller. Windows server has an easy-to-use UI and documentation is readily available should a problem be encountered during the deployment. | Hosting a web-based software application on a mobile device is a very poor idea for several reasons. without access to a stable wired ethernet connection and lacking the necessary hardware to perform the necessary tasks of a dedicated app server. It would not be ideal to attempt to host server-side code this way.  Project 2: I will stand by my previous opinion that using a mobile device to host a REST service, though possible, is not ideal and could not stand up to the rigorous hardware requirements and network latency that large companies depend on to hand their REST services. |
| **Client Side** | Since this application is totally web based the client OS is nowhere near as important as browser compatibility, because of this the project should be extremely cost effective because code can be written once and will be able to run on all platforms. The current project is started in Java, but as of today .Net 5 is in my opinion the best and most modern router to make an application that will run on Windows, Mac, and Linux with support for most smart phone browsers.  Project 2:  After working on the web-based client application for the Game Room and having the opportunity to use Java to code the server-side REST service. I would continue to use Java on the backend, and consider integrating a popular JavaScript framework like Vue, or React for the client-side web application. I was very intrigued by using Maven to handle dependencies and found that Drop wizard made handling all the libraries and their version control a breeze. Though I have limited experience working with JavaScript frameworks, they seem to be the ideal choice for web-based client applications. After some research while working on the server application I believe I could use Java Jackson library to handle the parsing of JSON data to handle the transfer of data from the client to the server. | Since this application is totally web based the client OS is nowhere near as important as browser compatibility, because of this the project should be extremely cost effective because code can be written once and will be able to run on all platforms. The current project is started in Java, but as of today .Net 5 is in my opinion the best and most modern router to make an application that will run on Windows, Mac, and Linux with support for most smart phone browsers.  Project 2:  After working on the web-based client application for the Game Room and having the opportunity to use Java to code the server-side REST service. I would continue to use Java on the backend, and consider integrating a popular JavaScript framework like Vue, or React for the client-side web application.  If a desktop application was preferred, even though the web-based application is available, using Java for the client-side UI would be an easy way to integrate the software to any OS that has the JVM installed and would be an easy integration with the Java based REST Service already in use. | | Since this application is totally web based the client OS is nowhere near as important as browser compatibility, because of this the project should be extremely cost effective because code can be written once and will be able to run on all platforms. The current project is started in Java, but as of today .Net 5 is in my opinion the best and most modern router to make an application that will run on Windows, Mac, and Linux with support for most smart phone browsers.  Project 2:  After working on the web-based client application for the Game Room and having the opportunity to use Java to code the server-side REST service. I would continue to use Java on the backend, and consider integrating a popular JavaScript framework like Vue, or React for the client-side web application.  If Java was used to write the client-side code on the mobile application, much of the code be reused for a desktop application on all three popular OS. | Since this application is totally web based the client OS is nowhere near as important as browser compatibility, because of this the project should be extremely cost effective because code can be written once and will be able to run on all platforms. The current project is started in Java, but as of today .Net 5 is in my opinion the best and most modern router to make an application that will run on Windows, Mac, and Linux with support for most smart phone browsers.  Project 2:  A Java based mobile application would be an excellent choice for the client-side mobile application. Much of this code could be reused if a future desktop application was need and because the JVM allows a “universal” usability it would be any easy and cost-effective method to handle creating a client-side application that could run on multiple operating systems. After using java libraries like Jackson to parse JSON data, I have gained a better understanding of the communication between the client and server. I would continue to use the existing libraries that are in place to facilitate the transfer of data via HTTP through the REST service. |
| **Development Tools** | The relevant programming languages and tools are the same for each operating platform, If going the .Net 5 C# route, all code can be written in C# with some front end hybrid code called Razor that effectively combines C# with Javascript, HTML, and CSS. The programmer would need access to Microsoft Visual Studio 2019, experience with Microsoft SQL Server, and an application like Postman to make easy API calls. Deployment is easy and can be handled quickly with Microsoft Azure to host the application in a cloud based server.  Project 2:  After creating both the server application and web-based client application. There are multiple changes I would make to the development tools being utilized to complete this project. I have found Java, Maven, and Dropwizard to be a fantastic way to handle dependencies on the server side application. I would also user a different IDE like eclipse that excels in Java development particularly. Since Dropwizard makes it easy and creates a project with an existing maven pom.xml file with the required dependencies, the library repositories also come included and that means eclipse uses these repositories for code completion. This is a great way to speed up development and provides and easy way to read documentation on the various methods and classes of a library without having to sift through their online documentation. Like I mentioned earlier, the Jackson library makes parsing JSON data simple in Java. JSON has become the standard structure for sending and receiving data from REST services, so this is an important library to utilize. A benefit of Maven is I can simply edit the pom.xml file should I want to include any other dependencies that I would find useful. Another useful tool would be an application like Postman. Postman makes sending HTTP requests easy and provides an easy-to-use UI to see the sent data, values, and keys along with their responses. This is essential when testing the functionality of the REST service, and could save a lot of time over the duration of the project development. | | The relevant programming languages and tools are the same for each operating platform, If going the .Net 5 C# route, all code can be written in C# with some front end hybrid code called Razor that effectively combines C# with Javascript, HTML, and CSS. The programmer would need access to Microsoft Visual Studio 2019, experience with Microsoft SQL Server, and an application like Postman to make easy API calls. Deployment is easy and can be handled quickly with Microsoft Azure to host the application in a cloud based server. | The relevant programming languages and tools are the same for each operating platform, If going the .Net 5 C# route, all code can be written in C# with some front end hybrid code called Razor that effectively combines C# with Javascript, HTML, and CSS. The programmer would need access to Microsoft Visual Studio 2019, experience with Microsoft SQL Server, and an application like Postman to make easy API calls. Deployment is easy and can be handled quickly with Microsoft Azure to host the application in a cloud based server. | The relevant programming languages and tools are the same for each operating platform, If going the .Net 5 C# route, all code can be written in C# with some front end hybrid code called Razor that effectively combines C# with Javascript, HTML, and CSS. The programmer would need access to Microsoft Visual Studio 2019, experience with Microsoft SQL Server, and an application like Postman to make easy API calls. Deployment is easy and can be handled quickly with Microsoft Azure to host the application in a cloud based server. |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

**Operating Platform**: I would recommend The Gaming Room, utilize a Windows Server to host a Java application using Dropwizard and its packaged libraries. With native support for Windows, Mac, and Linux, should there be a need to change the OS of the server a packaged .JAR file can be easily installed and ran on the server. With a multiple server architecture, it would be easy to have exact copies of the server software on each region’s server. An additional server may also be added per region as a dedicated database server should migration of the current database be necessary. I Would also recommend that the server application run on a cloud-based server or multiple depending on number of clients and the targeted latency to the client application in its communications with the server. By utilizing Java again for the client application, we are afforded the ability to write the client application once and wrap the client for each of its intended platforms. Java will run natively on Windows, Mac, and Linux desktops but can be altered to run on IOS and Android without the need to rewrite the entire client application. This is a great way to offer The Gaming Rooms software to many diverse users and save costs by decreasing development time writing client applications individually for each targeted platform.

1. **Operating Systems Architectures**: Windows Server 2019 and Microsoft SQL Server either locally hosted or in the cloud for lower latency depending on client locations. The client server could be hosted on a virtual machine on the current application server should it have the required hardware to support this functionality and should there be no conflicting port forwarding issues. If the need arises for multiple servers, by utilizing a .jar file for the server application, new servers can be added with ease and which server is used can be handle by a region selection on the client application.
2. **Storage Management**: The appropriate storage management solution would be to either continue utilization of the current database with support for the new web-based application or start a migration of the current database to one that would support functionality for the web-based application and the current mobile application.

Additionally, upon further thought of the storage management requirements to successfully implement the Game room client and server applications it is important to note the need of protecting the data the is housed on the server and any subsequent database servers. I would recommend a RAID0 configuration of the OS disk to keep a swappable backup of the OS platform and its configuration in the event of any catastrophic failure. It is equally as important to protect the data housed in the database from catastrophic failure or any malicious attempts to access the data by unauthorized sources. A RAID5 configuration would work well to reduce strain on any single disk in the event a drive does fail. The benefit of RAID5 is there will always be a spare handy already in place. It is important that confidential data is encrypted in the database as well as to and from the client that is authenticating to ensure the security of the client’s private account. To maintain speed, any non-secure information traveling from the server can be sent via UDP without encryption to improve latency and allow our users a smooth experience utilizing the client application.

Storage requirements of the client application will be minimal and all errors due to insufficient storage space should be handled by the client OS. Though it is important careful attention is paid to ensure our software is saving information properly on the client hardware and data is not being damaged during transfer.

1. **Memory Management**: Memory management though handle natively through Java; it is important to pay close attention to wasting memory on the client and server applications respectively. With only a finite amount of RAM on each platform it is important to keep the client application usage as low as possible to preserve space on the client’s hardware for any additional applications or functions of their hardware. On the server, it is important to utilize as much memory as possible without ever reaching the maximum which could result in downtime or poor performance for clients trying to contact the server.
2. **Distributed Systems and Networks**: The web-based application and the current mobile application need access to the same database information and may require asynchronous functionality to await changes to the database with the influx of new users accessing the data from various platforms. In terms of network connectivity hosting the new web application in a cloud-based server will offer better security in the scenario of outages. In the multiple server architecture, should one server go down, and there is ample room to handle additional clients from differing regions the application can be configured to pick up on clients whose main server may be unavailable.
3. **Security**: The web-based application will require user authentication when accessing the application and making changes to data stored in the database. Additionally, data stored in the database should be encrypted when it is being stored and during transit of the packets. Should communication between the current mobile application server and database require a network connection to the new web-based application server, it should only be through a secure site to site VPN tunnel. Authorization should be handled through the transaction of a secure token system that is passed via HTTP response and cached on the client hardware after signing in, should the client become unresponsive for an extended period the client will be signed out and need to authenticate again with the server to receive a new authorization token.