

CS 230 Project Two Game App

# **CS 230 Project Software Design Template**

Version 3.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 3.0 | 2/17/2023 | Evan Arthur | Expansion of recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

Client: The Gaming Room

Market: Gaming

The client is seeking to develop a web-based version of a game currently available only as an Android app. The web-based version should reflect the apps functionality as closely as possible.

## Requirements

Software Requirements:

* 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.

Business Requirements:

* Remain within the specified budget.
* Meet the specified deadline.

## [Design Constraints](#_2et92p0)

* Multiple operating systems to account for which may lead to increased cost and time for development.
* Budget may affect the complexity of the software and the timeframe.
* Remaining true to the current Android app may cause difficulties when developing for operating systems such as MacOS.

## [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)

Entity is a superclass creating a relationship between Game, Team, and Player. Each of the subclasses inherit information from Entity. If we look further, we can see the aggregation among the classes with GameService referencing Game, Game referencing Team, and Team referencing Player.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | * Flexible terminal * Upgradeable * Multiple options for web hosting requirements * Less preferred among other options | * Flexible terminal * High security * More preferred among other options * Difficult to develop | * Most software among other OS * Closed platform * Most resources available * Most susceptible to viruses | * Most popular * More compatible * Poor security |
| **Client Side** | * Moderate expertise * Moderate cost | * Max expertise * Minimum cost | * Minimum expertise * Moderate cost | Multiple platforms among mobile devices which could increase expertise and cost requirements |
| **Development Tools** | * Swift is the most popular tool * Notepad++ * Able to run all languages * Most popular libraries to support frontend | * Visual studio * Eclipse * Notepad++ * Able to run all languages * Most popular libraries to support frontend | * Visual studio * Eclipse * Notepad++ * Able to run all languages * Most popular libraries to support frontend | * Android * Swift * Able to run most popular languages * Most popular libraries to support frontend |

## The above chart gives a broad idea of some of the benefits as shortcomings of the different platforms. We will elaborate further here to ensure a better understanding of the potential for each option. Starting with the Server Side of things, All the platforms offer a server-based deployment method that can be used in this application. As for costs related to these options, Linux is open source thus making it the most cost effective option but in the same regards there is less support due to it being open source. The other three options are proprietary and are less cost-effective options but the support behind them is greater. If cost is the main priority, Linux would be the best option but it could lead to other unexpected expenses in the future with the lack of support.

Moving on to the Client Side of things we will look at cost first. Linux is again going to be the most cost effective but may require a high level of expertise among the other options which could eliminate its cost effectiveness. The increase level of expertise required may also make development more time consuming and push back the release date of everything. Mobile devices may run into similar issues as Linux as there are multiple platforms to consider such as IOS and Android. The other two options have a lower level of expertise required but may have a higher up-front cost.

As far as Development Tools go, there is enough popular and good options available across every platform that it is almost unnecessary to discuss at length. While there may be some that are slightly better than others for each platform, it would not be difficult to choose one singular option that would work successfully. One option to consider is using Swift for MacOS and IOS which would not be a great option for Linux or Windows. But there are enough other options that would work well enough on MacOS and IOS as well as Windows and Linux that Swift should not be necessary.

## Recommendations

1. **Operating Platform**: While Linux would be the cheapest cost upfront, Windows would be the best platform to start with as it would be the most cost-effective long term with the future support and services, and it would require the least amount of expertise. It also provides more than enough software and IDE’s to be successful.
2. **Operating Systems Architectures**: Windows provides services used across all Windows applications. This allows Windows to provide a Graphical User Interface while accessing system resources among other things. Windows is built as a hybrid architecture using kernels, microkernels and layers to provide a comfortable balance of speed and security among other things. Windows includes a layer called the hardware abstraction layer which acts like a buffer between the hardware and software to provide a consistent experience across different hardware.
3. **Storage Management**: Windows 10 provides a feature called storage sense which allows the various options for managing files on the hard drive. These options include, among many other abilities, limiting space usage, specifying locations for apps, and the option for cloud storage.
4. **Memory Management**: Alongside the game, a library of pictures is also needed. Allocating memory for the library of pictures makes for an easy and succinct storage that won’t be affected when working on the code of the game. The total storage for the pictures and game is not large and the necessary information for each round could easily be pulled into the RAM in between each round for quick access. Any form of memory management a device uses should meet the requirements for the application.
5. **Distributed Systems and Networks**: After doing some research, I have found multiple options for creating a cross platform application. Some of the more popular options include Ionic, React Native, and Flutter. Based on the requirements for this application I would advise Ionic be used at it appears to be the best suited for a web-based design. If cross platform was not the intended route, Windows is still the most popular operating system among household computers as it is preinstalled in nearly every computer that is not from Apple.
6. **Security**: Windows comes with security software that is already set in place. It would be recommended to include another source to add another layer. It is never a bad idea to add layers of security. A breech of security can be very costly to resolve and can have a devasting effect in public confidence. With this being said, adding a user login that includes a unique username, email address, and password will provide further security from the users side as well as help monitor and eliminate undesired activity among users. At the current time since there is no extremely sensitive information necessary from any user (i.e. cc number, ss number, birth date) this should be sufficient security.