

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.0

## 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**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | 01/26/2024 | Richard Wellman | Updates: Executive Summary, Requirements, Design Constraints, System Architecture View, Domain Model, Evaluation and Recommendations. |
| Version | Date | Author | Comments |
| 1.1 | 02/11/2024 | Richard Wellman | Updates: Added to design constraints, added to Evaluation - notably all columns for Server, Client, and Devlopment Tools |
| Version | Date | Author | Comments |
| 1.2 | 02/20/2024 | Richard Wellman | Updates: Deleted Instructions, Added to Operating Platform, Operating Systems Architecture, Storage Management, Memory Management, Distributed Systems and Networks, and Security |

**Executive Summary**

The Gaming Room company wants to create a new cross-platform web-based game so as many players can play as possible, currently they only have an Android app of their game "Draw It or Lose It." A web-based game will eliminate the need to write native code for each platform. A web-based game played in a browser will allow cross-platform support and updates to be easily deployed. The only other operating system to consider is the mobile market. A separate application must be made for iOS to address those users.

**Requirements**

* A game can 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.

**Design Constraints**

* Must be a web-based game
* Must work in all web browsers across multiple platforms
* Teams consist of multiple players
* A check to ensure teams have enough players
* Database needed to save names and to check against to assure unique names
* Security measures need to be considered to protect user data

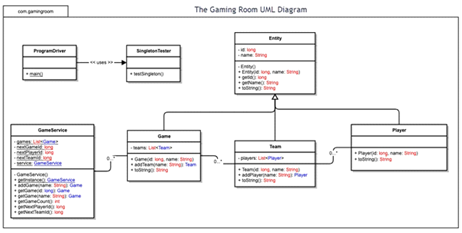
**System Architecture View**

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**

The Program Driver uses the Single Tester class to make sure there is only one instance of the class GameService. This is also indicated by the Black arrow with the symbol <<uses>>, indicating a one way and one time relationship. The GameService class can contain many or no game instances, thus ‘the zero to many’ symbol in the UML diagram. The entity class is shown to have aggregation as it is used by the Game class, Team class, and Player class. The Game to Team and Team to Player association lines all use a ‘zero to many’ symbol, indicating there can be ‘zero to many’ Game, Team, and Player class instances.

In more concise terms, the Game class uses the GameService class to be instantiated or to 'exist' and then the Game class has a list of Team instances (objects), and the team class has a list of Player instances (objects), and finally we have the Player instance themselves, which contains a player ID and a name. The Team has an ID, as does the Game itself, and the GamService can only be instanced or 'exit' once. Allowing the use of an Entity as a parent class keeps us from having to re-write the methods that are needed by all three classes, such as getID(). This invokes the principle of DRY or 'Don't Repeat Yourself.' This makes the code clearer and easier to read and to manage as the code base grows.



**Evaluation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Server Side** | **//As a note: From Apple Support directly: As of April 21, 2022, Apple has discontinued macOS Server.//**  **Characteristics:**  Apple Mac OS Server offers a more user-friendly approach to web services. Proprietary and maintained by Apple.  **Advantages:**  Mac as a server-side OS is useful with other Mac devices, seamless integration with said devices and Apple's proprietary software due to Apple's platform.  User-friendly interface and built-in security features.  **Disadvantages:** Limited vendor support (like Linux), limited scalability for large enterprises, limited support of virtualization and cloud computing, Apple has discontinued server support, so web server support will not be an option.  **Update:**  Server side can be handled through cloud computing. PHP, Python, Ruby, C#, and JavaScript (Node.JS), are server-side languages that can be used to write code for the back- end. Since the hardware is handled by a third-party vendor, a trusted service would be carefully considered though virtual, or cloud computing allows for robust security and containerization. We pay only for what we need and that keeps costs to a minimum. | **Characteristics:** Linux is an open-source OS used by most of the world’s servers and available to PCs as different distros.  **Advantages:** Linux is an open-source OS and as such the community helps to look for security threats, and as in house server OS, Linux can be customized to fit business needs and it is free being open-source. **Disadvantages:** limited vendor support for web hosting services and as such in-house developers are needed.  **Update:**  Server side can be handled through cloud computing. PHP, Python, Ruby, C#, and JavaScript (NodeJS) are server-side languages that can be used to write code for the back- end. Since the hardware is handled by a third-party vendor, a trusted service would be carefully considered though virtual, or cloud computing allows for robust security and containerization. We pay only for what we need and that keeps costs to a minimum. | **Characteristics:** Widely supported by vendors, so much easier to find applications that work well with Windows Servers.  **Advantages:**  Wide array of vendor support, virtualization, and cloud support, easily scalable for large enterprises  Active Directory integration and authentication.  **Disadvantages:**  Security issues such as hackers love to target Windows platforms, constant updates necessary to stay ahead of attacks with many Unknown/unknowns, or zero-day attacks to be concerned with.  **Update:**  Server side can be handled through cloud computing. PHP, Python, Ruby, C#, and JavaScript (NodeJS) are server-side languages that can be used to write code for the back- end. Since the hardware is handled by a third-party vendor, a trusted service would be carefully considered though virtual, or cloud computing allows for robust security and containerization. We pay only for what we need and that keeps costs to a minimum. | **Characteristics:**  Mobile devices used as a web server are uncommon but possible. You need specific tools to make this happen, it is not capable (or intended to be) by the developers, except Nokia.  **Advantages:**  Physically mobile, so you can take your server anywhere you need or want.  **Disadvantages:**  Serious security concerns, Bluetooth, Wi-Fi, etc. all offer effortless ways for a hacker. Limited storage size, and limited performance as devices like Android OS based phones are not intended to be web servers, and so do not offer the same speed or reliability a server (traditional) would.  **Update:**  Server side can be handled through cloud computing. PHP, Python, Ruby, C#, and JavaScript (NodeJS) are server-side languages that can be used to write code for the back- end. Since the hardware is handled by a third-party vendor, a trusted service would be carefully considered though virtual, or cloud computing allows for robust security and containerization. We pay only for what we need and that keeps costs to a minimum. |
| **Client Side** | Mac OS can run any software assuming it has the correct compiler or interpreter associated with it. iOS uses Swift (a derivative of Objective C) for coding and is approximately 20% of the mobile market. Those customers (iOS) tend to spend more than Android users in the store, so it is worth considering. Also, there is an annual cost for the Apple Store and developers that specialize in iOS cost quite a bit to employ.  **Update:**  RESTful APIs allow for secure communication and are easy to implement thus keeping costs down. RESTful can be integrated within the app and used cross-platform so no extra work would need be done for each platform, other than writing it in that environment, specifically Android and iOS. For a website, all three major platforms are covered, though the Android and iOS app would need to be natively written, despite being able to run a browser on mobile. So, Swift or Objective C code to handle the client-server RESTful APIs vs JavaScript in the other three platforms. This allows for stronger security as the web application has the same code amongst all three platforms, Mac, Linux, Windows and as such whatever updates occur to the web app, occur for all platforms. | In-house support would be necessary for Linux based applications. No cost as it is open source, but it is in fact still costly to write an application as developers are not free. Time to produce an app or service may be longer as there is not a lot of support from third party vendors, but the community is highly supportive. Again, in-house support would be necessary.  **Update:**  RESTful APIs allow for secure communication and are easy to implement thus keeping costs down. RESTful can be integrated within the app and used cross-platform so no extra work would need be done for each platform, other than writing it in that environment, specifically iOS. For a website, all three major platforms are covered, though the Android and iOS app would need to be natively written, despite being able to run a browser on mobile. So, Swift or Objective C code to handle the client-server RESTful APIs vs JavaScript in the other three platforms. This allows for stronger security as the web application has the same code amongst all three platforms, Mac, Linux, Windows and as such whatever updates occur to the web app, occur for all platforms. | Windows is the most widely used OS (desktop) and thus has the most support for client-side applications. This could speed up development time, especially with frameworks like .NET/C#. Windows is its own environment much like Mac but is more widely used so it would be easier to find developers comfortable with Windows than any other OS. If a Windows application is needed, C#/.NET is the way to go.  **Update:**  RESTful APIs allow for secure communication and are easy to implement thus keeping costs down. RESTful can be integrated within the app and used cross-platform so no extra work would need be done for each platform, other than writing it in that environment, specifically Android and iOS. For a website, all three major platforms are covered, though the Android app would need to be natively Android, despite being able to run a browser on mobile. So, Swift or Objective C code to handle the client-server RESTful APIs vs JavaScript in the other three platforms. This allows for stronger security as the web application has the same code amongst all three platforms, Mac, Linux, Windows and as such whatever updates occur to the web app, occur for all platforms. | Limited resources so high traffic could render a web server useless if it is more than a basic static website. If requests are constantly being sent to the server, it is likely to cause slowdowns to a crawl. Not as widely used, and thus not as much community support, but it is feasible for a small web app. It would be best to delegate the details to each phone (account information), though that is a security concern some kind of encryption would be necessary.  **Update:**  RESTful APIs allow for secure communication and are easy to implement thus keeping costs down. RESTful can be integrated within the app and used cross-platform so no extra work would need be done for each platform, other than writing it in that environment, specifically Android and iOS. For a website, all three major platforms are covered, though the Android and iOS would need to be natively Android, despite being able to run a browser on mobile. So, Swift or Objective C code to handle the client-server RESTful APIs vs JavaScript in the other three platforms. This allows for stronger security as the web application has the same code amongst all three platforms, Mac, Linux, Windows and as such whatever updates occur to the web app, occur for all platforms. |
| **Development Tools** | Specific to Mac OS there is Objective C and to iOS there is Swift, though MacOS can run any software with appropriate interpreters or compilers. For iOS however, Swift is the primary language now used. X Code is Apple's IDE for software dev, meant to target all Apple platforms. Mac is excluded from Windows and Linux though there are ways to 'hack' a Windows PC to make it work with Mac, though that is not legal and easier to just get a MacBook.  **Update:**  The IDE for Mac is Xcode, though since this is a website meant to run in a browser, VSC (Visual Studio Code) works well if this is to be the platform on which the website is built, and then can be run on any web browser. As previously mentioned, once developed using HTML, CSS, JS, and RESTful for back-end client-server communication, there is no need to worry about the underlying operating system as it is a web-based application. | Much like Mac OS, Linux can support many languages and again assuming the right compilers and interpreters are installed. Linux itself is written and C/Assembly and so is low-level and highly efficient. Assuming vendor support of any relevant IDEs, the list is massive but popular, including Eclipse (Java), IDLE (Python), and Android Studio (Java, Kotlin). Anything can be made in a Linux environment, except MacOS/iOS Xcode IDE, which tools are available as open source, but not the IDE itself.  **Update:**  Any IDE, except Xcode and Visual Studio can run on Linux. However, VSC (Visual Studio Code) can run on Linux, but again once the website is up and running, any user with a browser can run it. As previously mentioned, once developed using HTML, CSS, JS, and RESTful for back-end client-server communication, there is no need to worry about the underlying operating system as it is a web-based application. | Windows can support many languages, being the widest used OS (desktop) and third-party support is extremely high. IDEs include Visual Studio (C#/.NET), Eclipse (Java), IDLE (Python), etc. though many languages are supported by Visual Studio and thus can be used as the only IDE on Windows if wanted or needed. The community version is free of VS and there is a VSC (Visual Studio Code) which is lighter and can be run on Mac.  **Update:**  Excluding Xcode any IDE can run in Windows. As previously mentioned, once developed using HTML, CSS, JS, and RESTful for back-end client-server communication, there is no need to worry about the underlying operating system as it is a web-based application. | There are two primary languages for application development for mobile. It is really two separate environments and OS's, Apple iOS, and Android. The latter is an open-source, created by Google and open-sourced by them as well. iOS is needed for Apple phones and devices that do not run the Mac OS and Android runs on many devices (not just phones) and is increasing by the day with increase of IoT (internet of things).  **Update:**  Android Studio is the best option for mobile based development and currently Kotlin is recommended for the programming language though Java is still widely supported. A browser based game may have unwanted or unpredictable patterns or behaviors on a mobile platform (varying screen sizes as an example can cause bugs in the way the website is laid out), though some libraries allow for scaling, it is not fail proof in all situations and would take a considerable more amount of time to debug and fix, than to write a native Android application that can be updated on the Play Store and pushed to all customers at once. Thus, it is best to have a standalone Android application for mobile users. |

**Recommendations**

* **Operating Platform**: The most widely used of these platforms is Windows and Android. For a company starting out, a Windows application could be developed in Linux and Android applications could be made in Linux as well. Interestingly Android uses the Linux kernel as a base for its OS, and Linux is a safe choice in terms of security for a small business. If cloud computing is not used, Linux servers are a great option as the security is customizable and robust as only what is truly needed is installed (updates with Linux distros happen frequently). This allows for easier growth of the infrastructure, and the applications made in Linux could be transferred over to Windows IDEs, if necessary, as the IDE for Android is the same environment for Windows. Linux does offer a bit of a learning curve, but anyone familiar with Mac OS should be able to pick it up quickly, and anyone with Windows experience can learn it as well. Linux would be the most cost-effective solution for in-house servers and business computer operating systems.
* **Operating Systems Architectures**: The main components of the Linux OS consist of Application, Shell, Kernel, Hardware, and Utilities layers. The kernel is the base component of Linux it controls the behavior of hardware components. The Shell is the interface to the kernel, this is where commands are entered, and the kernel then carries out. In Linux, the Bash Shell is the most widely used shell, but there are others. The Hardware layer consists of the kernel functions, memory management, CPU control, and I/O operations. The Utility layer is a specific set of command line tools that allow the user to do things such as file management, system monitoring, network configuration, and user management. It is important to note the Bash Shell (and others) are command line interfaces and thus are text based. This is a more efficient way to search query files in a massive directory.
* **Storage Management: “**Draw It or Lose It” is to be a web-based application, in house servers are a possibility using Linux servers but with today's market, a smaller business would do well with a cloud service storage management. If in-house were chosen storage management is handled through the command line interface, the Bash Shell, or can be UI/UX through a file manager or 'point and click.' It is much more efficient to allow a script to handle file storage and management as the file count and or directories become large. Bash scripts can be used to automate processes such as this, and to keep track of the memory in question.
* **Memory Management**: Java handles memory management automatically. The need to use pointers, references, dereferences, is not needed in Java and is one of the benefits of the language, the data in the stack is automatically handled. Kotlin takes this a step further and allows for simple and easy to read code as well as the newly launched Jetpack Compose which allows for more intuitive design via code, more of a traditional OOP style – like that of Unity. RAM specifications are a non-issue as the user’s hardware, if sufficient to run a web-browser, will be enough to handle running the game as the servers will handle all game functionality other than basic inputs from the users. As a web-based application, the player's system would be used for runtime and the back end (account info, game updates, etc.) would be on the server. JavaScript is the most widely used web programming language and despite the naming convention, shares little with Java syntactically. Memory for the servers needs to meet audience use and as such, cloud services that allow for quick changes to ram allocation is a smart choice. Many businesses are moving to the cloud model, so that the business can focus on their software and not stress about infrastructure. Web-browsers handle memory management for web-based applications as well, further simplifying the process.
* **Distributed Systems and Networks**: The benefit of using cloud-based servers is growth and dependability. The availability depends on the tier and the service agreement. Higher tiers of cloud services are expensive but are highly redundant and try to keep the service available to the user base. If the player has the proper OS and an internet connection, anyone can play the game.
* **Security**: As previously mentioned, Linux is open-source and community driven so many users of the platform are also on the lookout for exploits. Updates are frequent with the Linux platform and only what you need to install is installed on the Linux platform. Compared to Windows, which is bundled with bloatware, Linux is lighter and faster. Though careful consideration needs to be taken when using it as a server. Even if using cloud services, it is still up to the company to keep aware of security measures. More specifically, security falls to the software development team during development and after. Making sure to sanitize inputs and keep important code from being client-side. The infrastructure and hardware are handled by the cloud servers, but security still falls on the company on the software side and how it accesses and handles server-side code.

**References**

Google. 2024. https://developer.android.com/jetpack

Microsoft. https://www.microsoft.com/en-us/

Linux. 2024. https://www.linux.org/

Lavieri, D. E. (27-04-2019). Hands-On Design Patterns with Java. [VitalSource Bookshelf version]. Retrieved from vbk://9781789809954

Abraham Silberschatz, Peter B. Galvin, Greg Gagne (07-2008). Operating System Concepts. https://learning.oreilly.com/library/view/operating-system- concepts/9780470128725/silb\_9780470128725\_oeb\_c01\_r1.html#h1