

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](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 9/13/23 | Jaden Knutson | Initial draft covering client’s basic requirements. |
| 1.1 | 9/14/23 | Jaden Knutson | Reworked and added to requirements and constraints. |
| 1.2 | 9/15/23 | Jaden Knutson | Worked on Domain Model and Evaluation |
| 1.3 | 9/16/23 | Jaden Knutson | Worked on Windows recommendations section |
| 1.4 | 9/17/23 | Jaden Knutson | Double checked words, structure, and presentation. |
| 1.5 | 09/28/23 | Jaden Knutson | Added to operating system evaluation. |
| 1.6 | 10/13/23 | Jaden Knutson | Added Linux as primary recommendation. |
| 1.7 | 10/14/23 | Jaden Knutson | Added content to Linux recommendation. |
| 1.8 | 10/15/23 | Jaden Knutson | Finalize paper and reference section. |
| 1.9 | 10/16/23 | Jaden Knutson | Edited grammar and sentence structures. |

## [Executive Summary](#_sbfa50wo7nsh) - Introduce software design problem & present solution.

Creative Technology Solutions have partnered with The Gaming Room. The Gaming Room is a client looking to extend its current Android game app, “Draw it or Lose it”, into a muti-platform web-based game. This project’s goal is to streamline the game’s development process, and to achieve an easy transition from the mobile app to a web-based platform available across multiple operating systems and multiple platforms. For this project I am suggesting using Linux and keeping Windows as a viable second solution.

## Requirements – Section’s not being assessed but supports design constraints.

* **Multi-Platform web-based game** – The game will need to be adapted from an Android exclusive into a multi-platform app that is accessible on many platforms.
* **User/Game Data Storage** – The system must be able to manage, store, and retrieve different entities like players, teams, games, and game states.
* **Team-based gameplay** – Game must support one or more teams during gameplay which will allow teams to compete.
* **Player Assignment** – Each team must have the ability to assign multiple players to it, forming a team.
* **Unique Game/Team names** – To prevent naming conflicts, all game names must be unique along with any team or player names inside that specific game. This will then allow users to check specific name availability when they are creating teams or choosing their name.
* **Single Game Instance** – Only one instance of the game can exist at any one point. We do this by assigning unique “identifiers” to each player, team, and instance of game.
* **Singleton Design Pattern –** To ensure that only one instance of a game can exist, a Singleton design pattern will be used to manage game instances.
* **Real-Time Updates -** The game must provide real-time updates to all players without requiring a manual refresh. Updates for things like game events, scores, and changes.
* **User Authentication -** Users must be authenticated before joining a game to ensure a secure environment.
* **Security -** All user data and game interactions must be encrypted to protect against unwanted access and tampering.
* **Scalability -** The system must be able to handle an increasing number of users and games without affecting the performance.
* **Cost-effective -** The system must be developed/maintained within a budget without sacrificing quality or performance of the game.
* **Error Handling -** The system must handle errors for any unforeseen issues during gameplay.

## [Design Constraints](#_2et92p0) - Identify design constraints for developing the game.

* **Multi-Platform web-based game** – Expansion of the game to work on multiple platforms will require more development resources and will extend the time needed to ensure a good launch of the web-based game.
* **Team-based gameplay** – Team based gameplay requires tackling issues like user interface design or in game logic.
* **Player Assignment** – The ability for player team assignment will require a user-friendly interface for creating, adding, and managing team members.
* **Unique Game/Team Names:** Enforces uniqueness in game, team, and player names. A character limit might be imposed for names. Names that contain swear words, slurs, and other inappropriate naming schemes will have to be filtered or hidden.
* **Single Game Instance** – With a mandatory single game instance at one time, unique identifiers will have to be used, and this may influence aspects of storage and data management.
* **Browser Compatibility** – Must support a wide range of browsers like chrome, edge, firefox, etc…
* **Real-Time Updates -** Provides real-time updates to all players. Network latency could impact the speed of updates.
* **User Authentication -** Authenticates users prior to joining a game. There may be a rate limit on user authentication attempts.
* **Security -** Handles encryption of all user data and interactions. Extra system resources may be consumed due to encryption.
* **Scalability -** Needs to be designed for a growing number of users since it will be available on multiple platforms. Constraints include CPU, memory, or network limitations that may have to be reworked or upgraded in the future.
* **Cost-Effective** - Develop the game within a budget. Financial limits may dictate what technology is used, what features can all be included, and could possibly influence the efficiency, servers, and scalability of the game.
* **Error Handling -** Manages unforeseen issues during gameplay. This might limit functionalities during error scenarios.

## [Domain Model](#_8h2ehzxfam4o) - Describe the UML class diagram provided below.

* **ProgramDriver** – Top level class. Serves as the entry point of the application and contains the main() method. Carries out execution of the game and manages the programs flow.
* **SingletonTester** – This is for testing the singleton pattern to ensure correct implementation and ensuring that only one GameService instance exists in the memory at any one time. This is used by the Program Driver.
* **Entity** – Base class that holds basic behaviors and attributes for other classes. Includes ID and name which is useful to other classes in the diagram. Provides foundation structure for classes like player, team, and game.
* **GameService** – Responsible for managing game operation and maintaining the list of games. Uses singleton pattern to ensure only a single GameService instance is in memory. Is Associated with the Game class with a “0…\*” association, which means it can manage multiple Game instances.
* **Game** – Represents individual game session within the game. Inherits from the entity class. Associated with Team class with a “0…\*” relationship, which means it can include multiple teams. Manages game state along with included teams.
* **Team** – Defines the teams and represents groups of players within the game. Inherits from the entity class. Associated with Player class with a “0…\*” relationship meaning that a team can be made up of multiple players.
* **Player** – Defines individual players within the game. Inherits from the entity class. Encapsulates specific player attributes and behaviors allowing players to have a unique name and id in the game.

**"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) - Characteristics, advantages, & weaknesses of each operating platform

| **Development Requirement** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac servers are powerful and are very capable of handling a web-based game such as this.  However, these servers can often be more expensive than other competitors like Linux or Windows.  Mac servers offer great security along with good performance and integration with other Apple products.  Mac servers have limited software compatibility compared to the others.  Mac servers may not always be the best when it comes to aspects such as scalability. | Linux is one of the most budget-friendly and cost-effective choices for hosting a web-based game.  Some of the benefits of Linux include a wide range of open-source software, high scalability, extensive community support, and customization options like kernel tuning.  Linux has a good amount of security with its server due to the developer community frequently inspecting and adding to its codebase. This can help with patches and early detection of security risks.  The depth of customization is often a plus side, but it can also lead to possible security breaches if not handled appropriately.  Linux calls for specialized knowledge and is best used by experts in server management, which may add to the cost or amount of readily available developers with the needed expertise. | Windows servers are among the most straight forward options for setting up and hosting web-based games.  Windows servers offer extensive software compatibility, receive frequent updates, and are known for their ease of setup.  While Windows has a long history of being targeted by malware and viruses, Windows has taken an aggressive stance to heighten their security, especially among their servers.  Windows servers can be less customizable than Linux, and Windows servers come with licensing fees which is a drawback in terms of cost. | Mobile devices such as smartphones and tablets are not the best choice for hosting a web-based game.  Mobile devices come with a lot of limitations such as network constraints, data limitations, limited hardware capabilities, and limited battery life.  Overall, mobile devices are best suited for client-side use.  ­­­­ |
| **Client Side** | Developing client apps for the Mac universe will require expertise in macOS development and may end up being seen as costly compared to the other development options.  Mac universe users have a strong loyalty to their product, with this comes higher costs, but it also results in customers willing to pay higher for well-made applications.  The biggest issue then comes down to cross-platform compatibility where the mac universe can have some extra challenges and problems when trying to make applications available to other platforms. | Developing client apps for Linux is very cost effective.  Linux is an active open-source community that has many libraries and resources to aid in development.  Linux development may require special expertise and is not as straight forward as Windows or Mac.  This results in less user familiarity by using command-line tasks for installations and configurations. | Developing client apps for Windows offers great accessibility with a large user base to use as support.  Windows development comes with a wide range of software options and compatibility.  The familiarity of Windows UI makes it a popular choice with its greater user audience. | Creating client-side apps for mobile devices presents a couple considerations such as whether to release it on both iOS and Android markets.  The development of separate operating systems can require expertise in both fields.  Cross platform development options can come in handy for reusing code across different platforms. This may add a layer of complexity to the development process, possibly requiring more time.  Mobile devices also come with a bunch of hardware related and other types of limitations that may limit how expansive the development can be. Overall, the goal is to aim for optimizing user experiences across many different systems, which can be a challenge. |
| **Development Tools** | Software development within the Mac universe usually involves using Xcode as the main development environment.  Programming languages such as Swift and Objective-C are often used. | Software development for Linux usually involves IDEs like Eclipse or text editors like Vim.  Linux has support for multiple programming languages such as PHP, Ruby, and Python.  A lot of Linux development tool are open source, which is an advantage for cost and flexibility. | Software development for Windows often makes use of the Visual Studio IDE.  Visual Studio comes packed with usable programming languages such as c#, C++, .NET, ASP.NET, JavaScript, Python, SQL, and other languages and plugins.  Microsoft development tools often offer a vast amount of official support and documentation to aid developers. | Software development for mobile devices often uses Xcode for iOS, or Android Studio for Android applications.  Xcode for iOS provides a nice set of tools for testing, creating, and developing mobile applications for the Mac Universe. Programming languages such as Swift and Objective-C are often used.  Android Studio for Android provides a feature rich development environment for creation of apps for android devices. Programming languages such as Kotlin and Java are often used.  Cross-Platform frameworks are available to help streamline the process for making an app across both operating systems. Some of these frameworks are React Native, Flutter, and Xamarin. |

1. **Primary Recommendation – Linux – 10/13/23**

**Primary Recommendation – Linux**

**Linux:**

* Linux is the primary recommendation due to its open-source nature, flexibility, and impressive performance capabilities. It offers developers many tools and libraries, making it a cost-effective solution for businesses. The Linux open-source community consistently delivers updates, security patches, and a wealth of resources for optimization and troubleshooting.

**Operating Platform**:

**Linux Servers:**

* Linux servers are recommended for their reliability, security, and flexibility. Open-source variants like Ubuntu and CentOS are particularly suited for server environments due to their stability and performance. Given the cost-effectiveness and extensive community support for Linux, it stands out as the primary choice, provided there's access to experienced Linux developers.

**Operating Systems Architectures**:

**Linux Monolithic Kernel:**

* Choosing the right architecture for the game can influence the performance, security, scalability, and other essential factors. The Linux monolithic kernel is recommended for its high performance, stemming from its integrated and open-source nature. It can be customized and optimized to cater to the game's specific needs.

**Backend Framework Considerations:**

* For backend frameworks, we need to consider the necessity of real-time updates and scalability. We can also consider a single or hybrid framework depending on the requirements and expertise of the development team.
* A single framework can offer simplicity, consistency, performance, and a faster initial development speed.
* A dual or hybrid framework could use each one's strengths and leverage those to cover each other's weaknesses. However, this approach is much more complex in deployment. This results in a need for more communication, maintenance, and a longer development period.

**Node.js:**

* Given the game's requirement for real-time updates, Node.js, with its single-threaded, event-driven architecture, is ideal for a multiplayer game like this. This architecture ensures the server will be able to handle multiple requests at the same time without delays.
* A major strength of Node.js is its cross-platform capabilities, allowing it to run seamlessly across Windows, macOS, and Linux. This ensures compatibility while also simplifying deployment considerations.
* The npm ecosystem can significantly aid development with its wide range of libraries and tools.

**.NET Core:**

* If the team leans towards C#, C++, or a structured framework is desired, .NET Core stands out with its comprehensive features, security, and high performance.
* .NET Core provides a structured, comprehensive framework offering a wide range of built-in features, ensuring a streamlined development process.
* A strength of .NET Core is that it is highly known for its performance in scenarios requiring rapid processing and user input reactions.
* It contains security features and regular updates to ensure the game remains secure against security threats.
* It is also cross-platform, ensuring compatibility across different operating systems.

**Storage Management**:

* Choosing the correct storage management is essential for the game's storage of user data, game states, media files, and other needed data. Using a cloud service such as Amazon Web Services, Google Cloud, or Azure would make for excellent storage options that are both reliable and scalable. Cloud services such as these provide robust support for Linux-based applications.

* **Azure:** While Azure is a strong option, Amazon Web Services (AWS) and Google Cloud might have an edge with more specialized tools tailored for Linux deployments. If the business already has a Windows Server License, taking advantage of the Azure Hybrid Benefit may offer additional cost savings. Additionally, if using .NET Core, Azure may be the best option since they are both products developed by Windows. Azure Blob Storage can store large data amounts. Its integration with other Azure services ensures a seamless cloud experience. Its cross-platform capabilities are a significant advantage.

* **Amazon Web Services:** AWS, the world's leader in cloud storage, offers a comprehensive suite of services catering to various storage needs. Amazon Simple Storage Service (S3) provides a highly scalable object storage service to store the game's assets, user data, game states, etc. Amazon Relational Database Service (RDS) could be used for structural data and game states. AWL's global infrastructure provides low latency for accessing data, which is crucial for a real-time multiplayer game such as this. AWS also offers a suite for analytics, monitoring, and security tools, which help optimize the game's performance.

* **Google Cloud:** On the other hand, Google Cloud not only provides versatile storage solutions but also excels in AI and machine learning capabilities. This can be advantageous for game analytics, player behavior prediction, and enhancing in-game experiences. Google Cloud is well known for its high-performance computing, data storage, and machine learning capabilities. Google Cloud also offers unified object storage suitable for storing many of the game's assets, backups, and other large data sets. Google Cloud's global infrastructure ensures consistent, fast, and reliable access to the game's data.
* **Hybrid Strategy:** Combining cloud storage for unstructured data with MongoDB for structured data like user profiles and game states ensures a versatile storage approach that takes advantage of their strengths and reduces the impact of their weaknesses.

**MongoDB:** Recommended for Linux due to its scalability and adaptability. It's especially suited for evolving game requirements and can handle vast amounts of data. Its strong community is an added advantage.

* Overall, I suggest Google Cloud due to its AI compatibility along with a MongoDB hybrid structure to ensure an efficient storage solution that leverages the abilities of AI learning capabilities.

**Memory Management**:

* Linux's memory management capabilities are designed for high-performance gaming applications such as this. With built-in garbage collection mechanisms in backend frameworks like Node.js and .NET Core, Linux ensures efficient resource utilization with the frequent creation and destruction of short-lived game objects.
* The performance is further enhanced by caching tools such as Redis or Memcached, which store frequently accessed data like player profiles and game states in memory for rapid retrieval.
* Buffer management plays an important role in real-time multiplayer games, temporarily storing data during transfers to ensure quick processing and a seamless gaming experience.
* Memory pools optimize this process by preallocating chunks of memory, reducing the overhead associated with dynamic memory allocation.
* Monitoring system memory usage is essential to ensure optimal performance as the game scales. Tools like Valgrind are necessary to help developers detect memory leaks, analyze consumption patterns, and maintain efficient memory usage.

**Distributed Systems and Networks**:

* For a seamless and scalable gaming experience, a well-thought-out distributed systems and networks approach is crucial. Embracing a microservices architecture facilitated by Docker allows the game to be modularized into distinct, manageable services, such as user authentication and game state management. Kubernetes further complements this by overseeing load balancing and scaling, ensuring consistent performance across varying environments.
* Load balancers, such as NGINX or HAProxy, play a pivotal role in distributing incoming traffic across multiple game server instances, preventing any single server from being overwhelmed and ensuring uninterrupted gameplay.
* WebSockets, a real-time communication protocol that can be employed for instantaneous player interactions.
* To increase global reach and performance, cloud platforms like AWS or Google Cloud are recommended. These platforms offer scalability and provide a comprehensive suite of tools that can enhance the game's efficiency, availability, and overall user experience.

**Security**:

* Security is vital in gaming, especially when sensitive player data and in-game transactions are involved. It's essential to employ HTTPS to safeguard data transfers, ensuring end-to-end encryption and protecting against potential security threats.
* Implementing firewalls is the first line of defense, shielding the game's infrastructure from unauthorized access and potential breaches.
* With new threats emerging regularly, it's crucial to stay ahead by regularly updating and patching all software components. This proactive approach minimizes the risk of exploitation.
* Furthermore, services like Cloudflare or AWS Shield can be integrated to fight against increasing DDoS attacks targeting online platforms. These tools detect and mitigate threats, ensuring uninterrupted gameplay and maintaining player trust.

**Conclusion:**

* In conclusion, Linux is a top contender due to its open-source nature, cost-effectiveness, and vast community support. Its compatibility with various tools and platforms makes it a versatile choice for game development. However, the final decision should align with the game's specific requirements, the team's expertise, and the budget.

-------------------------------------------------------------------------------------------------------------------

## Secondary Recommendation – Windows – 9/16/23

**Secondary Recommendation – Windows**

* This secondary suggestion is to use Windows based on its expansive compatibility, ease of use, available software, broad support, and overall easy development process. We could also use Azure’s cloud service to handle many different aspects of our system and program. However, one problem that may arise is the cost as we scale up the program. We could also use one of the other primary cloud services, such as Amazon Web Services or Google Cloud Platform.

**Secondary Operating Platform Suggestion:**Windows as the operating platform.

* Windows Server offers a reliable environment that ensures availability and requires minimum expertise costs due to ease of learning and use. It’s also cost-effective, has tons of hardware and IDEs, has a user-friendly interface that makes development more straightforward, and is ideal for the potential growth and expansion of the game.

**Secondary Operating Systems Architectures Suggestion:**Windows NT kernel architecture.

* The NT kernel is pre-emptive and multi-threaded, has good isolation of running applications, ensures performance, reliability, security, compatibility, scalability, portability, and is an excellent choice for games requiring real-time updates with low latency.

**Secondary Storage Management:**

* For storage management, I suggest Azure Blob Storage as a cloud-based solution, Windows Server Storage Spaces for a non-cloud on-premise solution, or a Microsoft SQL Server. Since we are looking for future scalability, I will remove Windows Server Storage Spaces.
* For the other two, we could do hybrid storage where Azure Blob stores drawings and other media files related to the game and use Microsoft SQL Server for our more structural data like player, team, and game information.
* Azure Blob Storage is scalable, cost-effective, great for storing media, accessible from macOS and Linux, and has good security.
* Microsoft SQL server is scalable, great for storing structured data types, accessible from macOS and Linux using several libraries and drivers, has good security, and has automated performance tuning. It could be used on-premise or as an Azure SQL database.

**Secondary Memory Management Suggestion:** Windows Dynamic Memory.

* Automatically adjusts the amount of memory available to virtualized applications.
* Dynamically allocates memory, making for an efficient use of available resources.
* Helps improve performance by ensuring it has enough memory resources to run smoothly.
* With scaling, this efficient use of memory resources could reduce costs.

**Secondary Distributed Systems and Networks Suggestion:**

* Containerized architecture using Docker and Kubernetes.
* Windows supports Docker and Kubernetes, which can simplify and produce a scalable, distributed system.
* Uses containerization, which encapsulates dependencies, making deployment easier across different platforms.
* Azure Kubernetes Service provides additional ease of management and load balancing to distribute network traffic among many containers. This will improve the system’s performance and reliability.

**Secondary Security Suggestion:** Built-in Windows security features and Azure-based solutions.

* Windows Defender and Windows Firewall now offer reliable local security features, but also have a 3rd party security would be advised.
* Azure has many aspects, like Active Directory, for secure, scalable user authentication. Azure Key Vault has secure storage for sensitive user and application information. Azure offers encryption options, additional security measures, and a security center for advanced threat protection.

-------------------------------------------------------------------------------------------------------------------

**REFERENCES Part 1: (Evaluation)**

*Mac vs Windows: Advice for Web Developers*. (n.d.). Createdbycocoon.com. https://createdbycocoon.com/post/mac-or-windows-for-web-development

Cocca, G. (2022, April 12). *Windows vs MacOS vs Linux – Operating System Handbook*. FreeCodeCamp.org. https://www.freecodecamp.org/news/an-introduction-to-operating-systems/

*OS Showdown: Windows vs. macOS vs. Linux for Web Development*. (2019, July 10). Soshace. https://soshace.com/operating-systems-showdown-windows-vs-macos-vs-linux-for-web-development/

Gewirtz, D. (2019a, April 8). *Windows, Mac, or Linux? We compare the pros and cons of these computing platforms*. ZDNet; ZDNet. https://www.zdnet.com/article/windows-mac-or-linux-we-compare-the-pros-and-cons-of-these-computing-platforms/

**REFERENCES Part 2: (Recommendations)**

Wickramasinghe, S. (2021, October 1). *AWS vs Azure vs GCP: Complete Guide to Cloud Platforms for the Enterprise*. BMC Blogs. https://www.bmc.com/blogs/aws-vs-azure-vs-google-cloud-platforms/

‌Coursera. (2022, August 10). *What’s the Difference Between AWS vs. Azure vs. Google Cloud?* Coursera. https://www.coursera.org/articles/aws-vs-azure-vs-google-cloud

‌Wikipedia Contributors. (2019, September 14). *Linux*. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Linux

*.NET | Free. Cross-platform. Open Source.* (n.d.). Microsoft. https://dotnet.microsoft.com/en-us/

Desai, J. (2022, January 19). *.NET Core vs Node.js: What’s the Technical Difference?* Positiwise. https://positiwise.com/blog/net-core-vs-node-js#:~:text=NET%20Core

Patel, A. (2021, July 20). *Node.js vs .NET Core the winner?* C# Programming. https://medium.com/c-sharp-progarmming/node-js-vs-net-core-the-winner-af461e19c2f

Team, W. (2021, January 8). *NGINX and HAProxy: Testing User Experience in the Cloud*. NGINX. https://www.nginx.com/blog/nginx-and-haproxy-testing-user-experience-in-the-cloud/

*What is web socket and how it is different from the HTTP?* (2019, December 4). GeeksforGeeks. https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/#

*Kubernetes vs Docker: What’s the difference?* (2021, September 29). Dynatrace News. https://www.dynatrace.com/news/blog/kubernetes-vs-docker/

Molochko, A. (2023, May 15). *Why Is Linux a Monolithic Kernel? | Baeldung on Linux*. Www.baeldung.com. https://www.baeldung.com/linux/monolithic-kernel

*Valgrind*. (2020, July 22). Wikipedia. https://en.wikipedia.org/wiki/Valgrind

*Node.js*. (2020, March 13). Wikipedia. https://en.wikipedia.org/wiki/Node.js