

**Draw It or Lose It**

**CS 230 Project Software Design Template**

Version 1.0

**Table of Contents**

**Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | 08/10/23 | Tyler Logue | **I changed from using MacOs to the LinuxOS after doing some research with game cross platform. I added some information to Storage and Memory managenement that I thought could be critical.** |

**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**

I work for Creative Technology Solutions as Technolodgy Constultant. The company I work for has recently taken on a new client, called The Gaming Room. That wants to develop a web-based game that serves multiple platforms based on their current game, Draw It or Lose It and it is currenlty only available in the Andriod app store. The client has request specific software rquirements that must be met for the game application that I will go further in to detail later. They need our help in streamlining the development for the game application.

**Requirements**

The client's business requirements are that it needs to be a team based game that is loosely silimiar to the 1908s television game ***Win, Lose, or Draw***. Secondly they want to ensure in the game that it feels fresh, so we need to avoid duplication of games and names.

While the client's technical requirements are that there needs to a design pattern that verifies that only instance of the game exists in the memory by use of the singleton parttern to adapt an ordinary class. Also, we use iterator patterns to access elemnts of the game that are able to be used. Lastly, we need to ensure that the system can accomodate for large amounts of user growth.

**Design Constraints**

The design constraints for developing the game application in a web-based distributed enviroment is that security must be strong to protect users data and prevent malicous server attacks. As well the application should be programmed to run on ios and Andriod operating systems to help with growth. Lastly, the entity class works to create a realationship between Game, Team, and Player class

As for the implications of the design contstraints on application development we should incorporate the use of Handling real-time data synchronization to check that game data from different clients and servers remain up to date up-to-date.

**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 Gaming Room UML Diagram consists of seven classes: Entity,Game, Team, Player,ProgramDriver, SingletonTester, and GameService. Entity creates a realationship between the Game, Team, and Player class. We use the singleton pattern to force there to be only instance of the class created in the program to allow it to run effeciently. Each game can have multiple teams and each team can have multiple players. Game has an association with Team, GameService has an association with Game, Game has an association with team, and Team has an association with Player.



**Evaluation**

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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| **Server Side** | macOS is not as commonly used as Linux for server-side deployments. As well mac is a proprietary operating system which means the source code of the software is not available to the public, and only the company can make changes to its core codebase. | Linux is a popular choice, as it has strong security and is open-source. Also, the open-source nature means that there are no direct licensing costs to use the operating system. I would used server-side code languages either Node.js or Python. | Window OS is another viable option when it comes to hosting web-based applications. However, Windows has been known for security issues. | Mobile devices are nice to use for applications on the go and are mutl-platform generally. However, they can not act as the server for our game application as we want to ensure scalability, reliability, and security. This will happen by using |
| **Client Side** | You need to have a great knowledge of specific mac features such as Touch Bar. To reach cross platform then we would need someone is experienced in frameworks. Will have a different Operating System and game application. | This should be cost-effective as most of the tools are freely available to use. Will have a different Operating System and game application. Lastly, need use web technologies like HTML5, CSS3, and JavaScript to create the front-end of the game application. | This will be expensive like mac systems. However it is easy to learn and use. We would have to test across the different versions of windows to ensure compatibility, too. Will have a different Operating System and game application. | This also has developmental tool and IDE's that are generally free to help create programs. As well there are many operating systems when it comes to phones. So there will be more time needed for testing. Will have a different Operating System and game application. |
| **Development Tools** | programming languages: Swift, C, and C++.  IDE: XCode is used for macOS and iOS programming.  Other tools: Homebrew allows developers to install libraries. | programming languages: Javascript, Python,C and C++ are generally used.  IDE: Microsoft Visual studio code (that I personally use) and JetBrains is also used.  Cost: This should be cost-effective as most of the tools are freely available to use | programming languages: Javascript, Python,C and C++ are generally used.  IDE: Microsoft Visual studio code,Visual Studio Code, and JetBrains.  Cost: The expense for this can be high. | programming languages: C #,Java, Swift, and JavaScript.  IDE: Xcode for iOS and macOS.  Anroid studio for android app development. Visual Studio Code  Cost: |

**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**: Originally, I considered a Mac operating system but through research, I am going to choose Linux instead. As I believe the application has better security compared to WindowsOS and MacOS. This is due to its Unix-based systems having a robust user permission system. This is essential for gaming applications in this period of time due to all the data that can be extracted from microtransactions in games nowadays. Once the web-based game application and server-side code are finalized, I plan to deploy a Linux-based server. There are a number of options to choose from for our cloud hosting providers to host my server such as AWS, DigitalOcean, and Linode which will contribute to reliability and help ensure scalability.
* **Operating Systems Architectures**: The cool thing I learned about Linux is that the operating platform architecture is Unix-based and was designed to be portable. So that it may run on various hardware architectures with ease. Not only will this provide a stable and secure setting for the game application of "Draw It or Lose It", but it will be highly scaleable. This will help with the growing customer base that will inherently come from the new three upcoming operating systems that will be introduced.
* **Storage Management**: Recently, I discovered that I could use a database system such as MySQL can store game application data. SQL databases can be made scalable through sharding, effectively distributing the workload among different nodes. This approach significantly enhances response times and mitigates bottlenecks. It might be necessary to implement CDN (Content Delivery Networks) to distribute image assets across strategically positioned servers to improve the game applications' performance and scalability. Lastly, I found out that it is possible to implement asset sharding to distribute the workload across different nodes and servers.
* **Memory Management**: For LinuxOS the memory management techniques for the Draw It or Lose It game application will need dynamic memory allocation. This involves managing data structures, such as caches and user sessions, associated with ongoing gameplay. Linux's built-in cache systems that include object caching work to further optimize performance by storing frequently accessed data in memory, thereby reducing processing time. I discovered the LRU algorithm (Least Recently Used) can dynamically adjust the cache data from reviewing the users' gameplay."The idea behind the LRU algorithm is that items that have been used more recently are more likely to be used again in the near future, so they should be kept in the cache. On the other hand, items that have not been used for a long time are less likely to be used again, and therefore can be removed from the cache to make room for new items. (Testbook, 2023)".
* **Distributed Systems and Networks**: I did some research and found out that if I want to enable communication between various platforms and achieve scalability, I can deploy a distributed software architecture. This involves deploying different components of the game application on separate nodes running on different platforms. Distributed architectures allow you to reach different locations and data centers, ensuring responsiveness and scalability. You can add or remove nodes as needed to accommodate changes in demand. Dependencies between components should be managed carefully, and redundancy strategies should be in place to handle network connectivity issues or outages.
* **Security**: OAuth or OpenID connects with the use of APIs for the web application that will help to give specific roles and rules. API endpoints should be used to control access and permissions, with strong authentication mechanisms to prevent unauthorized access from accessing data or attacking the application. We will have mechanics such as a photo authenticator, and passwords that have a strength meter beside it. Lastly, I know that data encryption techniques are important as it secures the data from hackers when the client and server talk.

**Reference**

Loshin, P., & Bigelow, S. J. (2021). Linux operating system. Data Center. <https://www.techtarget.com/searchdatacenter/definition/Linux-operating-system#:~:text=Linux%20is%20a%20Unix-like%2C%20open%20source%20and%20community-developed,one%20of%20the%20most%20widely%20supported%20operating%20systems>.

Testbook. (2023). LRU Full Form: Check out what is Least Recently Used. Testbook. <https://testbook.com/full-form/lru-full-form#:~:text=The%20idea%20behind%20the%20LRU%20algorithm%20is%20that,the%20cache%20to%20make%20room%20for%20new%20items>.