

**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 | 01/28/24 | Eric Ross |  |

**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**

The Draw It or Lose It web version plans to offer a multi-platform gaming experience that retains the core gameplay of the original Android app. The design will allow for one or more teams to play simultaneously with each team having multiple players. The design will follow a singleton pattern to maintain a single game instance in memory. Doing so will ensure a unique game and team names that will be facilitated with unique identifiers for games, teams and players.

**Requirements**

The web-based version of the game should support multiple platforms.

Must render images in real-time and manage game rounds.

Must be able to have one or more teams with multiple teams members.

Must prevent name conflicts with unique identifiers for games, teams, and players.

Must use a singleton pattern to manage a game instance in memory.

Must have a consistent user experience across multiple devices and browsers.

**Design Constraints**

One design constraint for Draw It or Lose It includes designing a consistent user experience across different platforms. This will imply needing strong backend services. Another constraint will be managing the state of the game across distributed systems. This will require efficient database management for unique identifiers. Handling real-time interactions will be another design constraint. This implies the need for a front-end design compatible with different devices and browsers.

**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 UML diagram shows a structured domain model with the 'GameService' class as a singleton. This will make sure there is a single game instance. The 'Entity' class is a parent to the child 'Game', 'Team', and 'Player' classes. The 'Entity' class holds attributes and methods for the 'Game', 'Team', and 'Player' classes. The inheritace principle is shown indicated by the arrow from the children classes to the parent class with inheritance of the attributes and methods. The enscapulation principle is shown depicted in the 'GameService' class with private attributes. The principles help support an efficient design by reusing code and making sure the game entities are managed consistently.



**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.

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** | **Characteristics & Advantages:**  Unix-based, Professional developer tools, Compatible, Secure  **Weaknesses:**  Cost, Limited J2EE implementation  **Server-based Deployment:**  Tomcat can be easily installed. Can be integrated with Apache HTTP Server for enhanced performance and flexibility.  **Licensing Cost:**  It appears an unlimited client license can cost approximately $1000. Client could include the Apple Maintenance Program to receive any new OS X server versions released within 3 years of activation. | **Characteristics & Advantages:**  Open source, Highly customizable, Secure, Software compatibility, Cost, Stability, Performance  **Weaknesses:**  Prone to security threats although less common than other platforms, Performance issues with poorly configured systems  **Server-based Deployment:**  Linux can deploy web apps on physical servers, virtual machines, and cloud environments. Apache and Nginx web servers are known to run seamlessly on Linux.  **Licensing Cost:**  Linux distributions Ubuntu, Debian, and CentOS are free to use. Licensing costs are minimal or nonexistent | **Characteristics & Advantages:**  Consistent and flexible, Database friendly, Expanded server-side utilities, App support, Easy patch updates, Easy integration with Microsoft products and services  **Weaknesses:**  Cost, Reboot frequency for performance  **Server-based Deployment:**  Has network-based installation with Windows Deployment Services (WDS). Windows server licenses can be moved to Azure to save up to 40% costs.  **Licensing Cost:**  The Windows Datacenter Edition is ideal for highly virtualized datacenters and cloud environments. It can cost approximately $6,200. | **Characteristics & Advantages:**  Limited processing power, Less stable network connectivity, Limited memory and storage, Non-ideal scalability, Serve as an edge server, Mobile and flexible  **Weaknesses:**  Prone to battery drain, network interruptions, and hardware failures, Less security than a dedicated server, Scalability  **Server-based Deployment:**  Mobile devices are not typical for server-based deployment. They are better for client-side apps and edge computing.  **Licensing Cost:**  Mobile devices do not have direct licensing costs, though the cost will depend on the server OS. |
| **Client Side** | **Cost:**  The cost considerations of development may include purchasing hardware for testing, paying for software licenses or hiring Mac experts. Testing will have to take place to ensure cross-browser and mobile device compatibility.  **Time:**  The time considerations of development may be more than average due to platform-specific features, UI/UX design, and testing. Tests will be performed to ensure consistent behavior across browsers. Tests will be performed for mobile devices for platform-specific APIs and UI design.  **Expertise:**  The expertise considerations of development for the developers would include familiarity with macOS. They may also need to be familiar with programming languages such as Objective-C or Swift and also Apple’s development ecosystem. Developers need to understand HTML, CSS, and Javascript for web browser compatibility. For mobile device compatibility developers would need knowledge of mobile app development and platform-specific guidelines. | **Cost:**  The cost considerations of development would be attributed to the development tools, support, and maintenance.  **Time:**  The time considerations of development may vary based on the familiarity of Linux development and the complexity of the application. Thorough testing on different Linux flavors for compatibility for any platform-specific issues would be needed.  **Expertise:**  The expertise considerations of development for developers would include familiarity of Linux development tools, libraries, and APIs. The understanding of different Linux distributions such as Ubuntu, Fedora, and CentOS is very important for development. Developers will need to be familiar with HTML, CSS, and JavaScript to ensure browser compatibility. They would also have to be familiar with CSS for mobile device develpoment. | **Cost:**  The cost considerations of development would include the licensing fees for tools and libraries. Also, Windows developers may need to be hired. The web development should be cost effective due to open-source tools and cross-platform libraries. On the other hand, mobile development may be costly due to platform-specific requirements for iOS and Android.  **Time:**  The time considerations of development would include the testing required for Windows specific features. Testing would also include ensuring compatibility across various browsers and making adjustments. Additional time will be involved for testing the compatibility with the different mobile platforms.  **Expertise:**  The expertise considerations of development for the developers would include being well-versed in Windows frameworks such as UWP, WinForms, and WPF for example. Developers would also need to be familiar with HTML, CSS, and JavaScript for web browser compatibility. On mobile devices, developers will need knowledge of Swift or Objective-C for iOS and will need to know Java or Kotlin for Android. | **Cost:**  The cost considerations of development should be less than average since we should mainly focus on iOS as the game has already been developed for Android.  **Time:**  The time considerations of development should also be less than average since the bulk of the time is anticipated to focus on iOS. Although, additional testing may be needed to address any compatibility issues.  **Expertise:**  The expertise considerations of development for developers would include experience with iOS and Android systems. The developers will also need knowledge of HTML, CSS, and JavaScript for the mobile development. |
| **Development Tools** | A relevant programming language could be Java since it is beginner friendly and has many open-source resources. It is very useful in cross-platform development. JavaScript may also prove useful as it pairs well with HTML and CSS which makes it versatile with multiple platforms. Development tools such as Adobe AIR and GDevelop may be useful. Adobe allows for game creation across various platforms while GDevelop simplifies game creations for Mac. Different programming languages can impact the development team through skill diversity. Java developers can focus on the cross-platform aspect of the game while a JavaScript specialist will focus on the online components. Separate teams such as gameplay, graphics, and audio may be required for a large scale game but teams can work concurrently. Open-source tools like GDevelop can reduce licensing costs although some additional training may be needed. | A relevant programming language could be C++ as it is popular for game development due to its performance and versatility. While not as popular as C++, Java may be used with its framework LibGDX that allows developers to create cross-platform games. HTML5 and CSS are also viable for web-based games. The impact of collaboration on development teams will prove invaluable as effective communication is necessary for teams such as gameplay programmers, graphic artists, and sound designers. The separate teams need to be able to integrate different components seamlessly. There will be licensing costs with C++ and Java ranging from monthly to annual subscriptions. There is no licensing cost for using HTML5 and CSS. | A relevant programming language could be C# as it is popular for game development on the Unity game engine. It provides developers with a balance between performance and ease of use. JavaScript could also be used since it is used for web-based games and integrating game engines like Unity. Unity would prove useful as a game engine as it simplifies cross-platform development. GameMaker Studio could also be viable since it is ideal for 2D games. Godot is an open-source engine that provides support for 2D games as well. The impact of communication for the development teams is high as teams will need to coordinate effectively when working with different components such as graphics, gameplay, and audio. The teams will need a diverse skill set for working on different aspects such as the programming, art style, and sound design. The Unity game engine will come with licensing costs based on revenue or features provided. | Relevant programming languages would include HTML, CSS, and JavaScript. They are quite popular and allow for cross-platform game development. A diverse skill set will have a great impact on the development teams. Developers would need to be proficient in Java or Kotlin when developing for Android while developing for iOS would require knowledge using Objective-C or Swift. Multiple development teams will likely be needed for working with different operating systems. There may be some licensing costs if decided to use C# with the Unity game engine since it is popular for cross-platform games that support 2D development. |

**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 Linux as the operating platform for “Draw It or Lose It”. Linux is renowned for its performance and efficiency, especially in server environments. Linux is also considered secure by design. It has a strong permission system and active community support for security patches. Linux distributions are generally free, which can reduce the overall cost of operation. It also offers flexibility and control with extensive customization options to optimize server performance and resource use.
* **Operating Systems Architectures**: The kernel is the core of the Linux operating system. It acts as a bridge between the computer’s hardware and the application software. The kernel is responsible for managing the system’s resources, including memory, processing power, and I/O operations. Linux system libraries provide a set of standard routines and functions that application software can use to perform tasks without needing to interact directly with the kernel. These libraries abstract the complexity of interacting with the kernel and hardware, offering a more accessible programming interface. Linux system utilities are programs that perform individual, specialized management tasks. These tasks can range from file manipulation, system monitoring, network configuration, to user management. Linux supports Command Line Interface and Graphical User Interface options for user interactions. The CLI is powerful and efficient for system administration and scripting tasks, which are accessible through terminals like Bash. The GUI provides a friendly environment through desktop environments, which include a range of graphical tools and applications. The Linux shell is a command-line interpreter that allows users to interact with the kernel through a command prompt. The shell interprets user commands and scripts, invoking the appropriate system utilities or directly interacting with the kernel as needed.
* **Storage Management**: An appropriate storage management system to be used with Linux would be the XFS file system. XFS is known for its high performance, scalability, and reliability when dealing with large files and large volumes of data. It supports metadata journaling, which enhances data integrity and speeds up recovery in case of a system crash. XFS is particularly well-suited for applications that require high throughput for large files, making it a good choice for “Draw It or Lose It.” A network attached storage (NAS) system can also be advantageous. It allows multiple users and applications to access the storage over a network, making it suitable for collaborative environments and simplifying storage management. We can also utilize an object storage solution. Object storage is ideal for storing unstructured data like images, offering scalability, data durability, and ease of access, especially for web applications.
* **Memory Management**: Linux’s memory management system is designed to maximize performance and minimize latency. This is very important for interactive applications like “Draw It or Lose It”, especially when dealing with high-definition images and ensuring smooth gameplay. Linux uses virtual memory to extend the apparent amount of memory available to applications, using disk space to simulate additional RAM. This can allow “Draw It or Lose It” to utilize more memory than is physically available on the system, which is important for managing large image files and maintaining responsive gameplay. Linux also utilizes paging and swapping. Linux divides memory into pages and only loads the pages into memory that are currently required by running applications. This efficient use of memory ensures that the game can access and render images quickly without wasting memory on unused data. When the system runs low on physical memory, Linux can swap less frequently used data from RAM to a swap space on the disk. Doing so makes room for more immediate data needs of active applications. Linux also uses memory caching to store frequently accessed data in RAM, which reduces the need access slower storage mediums like hard drives or SSDs. This is particularly beneficial for “Draw It or Lose It”, where certain images or game assets might be accessed repeatedly. Caching those elements in memory speeds up their retrieval, enhancing game performance. Linux uses buffering to manage input and output operations efficiently, storing data in a buffer as it’s being transferred between two places. Utilizing this technique smooths out I/O operations for “Draw It or Lose It”, making sure that loading and rendering images does not cause stuttering or delays in gameplay. Linux provides dynamic memory allocation for applications. This allows “Draw It or Lose it” to request and free up memory as needed during its operation.
* **Distributed Systems and Networks**: For the distributed software we can apply a microservices architecture. Doing so will divide the game’s backend into microservices, each responsible for a certain aspect of the game’s functionality like user management, score tracking, and image rendering. This will allow for easier scaling, updates, and maintenance, as services can be developed, deployed, and scaled independently. We can also implement an API Gateway as the single entry point for all client requests. This routes requests to the appropriate microservice, handles load balancing, and provides an abstraction layer from the backend services. This will simplify client-side communication and provide a centralized location for concerns like authentication and rate limiting. For network connectivity and reliability we can use a Content Delivery Network. Using a CDN will cache and deliver static assets, like “Draw It or Lose It” images, closer to users worldwide. Doing so reduces latency, improves load times, and reduces the load on origin servers. Load balancers can be deployed to distribute incoming game traffic across multiple servers enhancing the game’s availability and reliability. Caching strategies can be implemented at various levels to reduce database load and improve response times. Also, using data replication across dispersed databases will ensure data availability and durability. When considering the dependencies between the components within the distributed systems and networks, Service Level Agreements can be defined for each microservice, specifying performance expectations and failure handling. This will help manage dependencies between service and make that the overall system remains robust. A circuit breaker pattern could be implemented to prevent a failing service from causing widespread system failure. The circuit breaker pattern temporarily halts requests to a service that is experiencing failures allowing it to recover and maintain system stability.
* **Security**: We can protect user information on and between various platforms within a Linux environment by securing data at rest, in transit, and enforcing strict access controls and monitoring practices. When securing data at rest, file system encryption can be utilized using Linux’s native encryption capabilities to encrypt disk partitions where user data is stored. This makes sure that data at rest is protected and inaccessible without the proper encryption key. Encryption can be applied within the database using features provided by the database management system. Doing so protects sensitive user information stored in the database. Strict file system permissions and access control lists (ACLs) can be implemented to restrict access to sensitive files and directories to only those services and users that absolutely need it. When securing data in transit, we can ensure that all data transmitted between the client applications and the servers are encrypted using Transport Layer Security (TLS). TLS is a cryptographic protocol that provides communication security over a network. For administrative access or inter-server communication, we can use a Secure Shell (SSH) with key-based authentication to secure the data in transit and prevent eavesdropping or man-in-the-middle attacks. Authentication mechanisms can be implemented for managing user access to the game. Multi-factor authentication (MFA) can also be considered to add an additional layer to security. Also, Role-Based Access Control (RBAC) can be used to define roles and permissions within the application to ensure users can only access data and perform actions appropriate to their role. Linux’s powerful firewall capabilities and cloud provider security groups can be utilized to control inbound and outbound traffic based on predefined security rules. We can keep the Linux operating system, software dependencies, and application libraries up-to-date with the latest security patches by conducting regular security audits of the system and application. Security Information and Event Management (SIEM) tools can be used to aggregate, analyze, and respond to security logs across the system. This will aid in detecting , investigating, and responding to security incidents more efficiently. A data backup and recovery plan should be implemented as well. Regularly testing backups will make sure they can be restored successfully. This will also minimize data loss in the event of a security breach or failure.