

*Draw It or Lose It*

**CS 230 Project Software Design Template**

Version 1.2

**Table of Contents**

**Document Revision History**

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/17/2022 | Aashish Shenoy | Design Template |
| 1.1 | 10/1/2022 | Aashish Shenoy | Expanded Dev Reqs |
| 1.2 | 10/14/2022 | Aashish Shenoy | Modified Recommendations |

**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 Gaming Room is looking to expand the development of its existing game, "*Draw It or Lose It."* Currently, the aforementioned game is limited to the Android mobile space, however, The Gaming Room (TGR), is seeking expansion into other platforms and operating systems. The game itself consists of a team guessing the puzzle which is rendered from stock images stored in a library.

**Design Constraints**

* The current Android only game needs to be expanded into a game that has the ability to function on multiple platforms.
* Effcient memory allocation will be required as each team will have teams, games, and players with unique names. Also needing only one active game instance in memory at a given time.
* Multi-user infastructure is needed to allow for multiple users to interact with game mechanics at the same time.
* All additions and adjustments must be functionally capable in a web-based medium.

**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**

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

We utilize a UML diagram to show the relationships between the various classes utilized in the game. The parent, Entitiy, class has the id and name properties, which are used by the Game, Team, and Player classes. The Team class can be associated with 0-to-many instances of the Player objects. This is similar in the cases of the Game class to Team objects, and GameService class to Game objects. The GameService class is the largest in terms of methods and houses framework for various methods. Fufilling one of the requirements of allocating only one game instance to memory at a time, the ProgramDriver class uses a SingletonTester class as a test.



**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** | An issue with MacOS would be it lack of discrete customization. One of the biggest advantages of MacOS the is security it offers. On the other side is the expense of iCloud hosting.  As of April 21, 2022, Apple has discontinued macOS. It alternatively did have an app for the price of 50$, however, Apple is transitioning out of it. Snow Leopard Servers are available for 499$. | Cheap and open-source. Two of its biggest advantages are compounded in other areas such as easily available hosting, increased security, and a huge range of available customizations.  Costs through Red Hat/IBM start from 349$ per year. | Notable licensing fees due to its popularity, and increased security risks when compared directly to Linux and MacOS. However, it does support MySQL which helps with web development.  Essentials Edition pricing is 501$, however, this increases to 1069$ for virtual environments and 16-core licensing. | Difficult, and needlessly expensive to run and maintain a web server from the confines of a mobile device. One option would be to use the Oracle Mobile Database, which would be very helpful during the initial start-up  Server costs vary significantly based on location and bandwidth requirements, most high end servers (given the expansion) will be around $100-$250/month. |
| **Client Side** | Since the game will be web-based the main requirement will be utilization and compatibility with Apple's main browser which is Safari.  The two main file structure systems for macOS include APFS (Apple File Systems) which is the default, and Mac OS Extended. The latter has four variations, but we can utilize Journaled HFS Plus for hierarchical file system protection.  Given the popularity, both cost and expertise level would be moderate. Exporting to a new platform would be the only time constraint given that the core game has already been developed. | Since the game will be web-based no significant Linux exclusive considerations need to be made to take into account. Simply allowing various browser compatability will allow the game to function as intended.  Given the popularity, both cost and expertise level would be moderate. Exporting to a new platform would be the only time constraint given that the core game has already been developed. | Similar to MacOS as well as Linux, using an existing browser will allow full functionality of the game. Expansion into Microsoft's preferred browser, Edge, could be possible.  All browsers such as Chrome, Edge, and Firefox have support for REST framework.  Given the popularity, both cost and expertise level would be moderate. Exporting to a new platform would be the only time constraint given that the core game has already been developed. | Given consideration of resolution and adjustment to the smaller screen sizes, simply utilizing the user's preferred mobile-based web browser would be sufficient enough to operate the game.  We will need a full stack web developer to formulate the infrastructure with our API. Also have the option to create a PWA (progressive web app) instead of access with just their website. An iOS specific developer would be needed as well.  Cost would be for this specific development might be higher than the other OS mentioned before. The expertise and time required would be determined if we would like to move forward with a native iOS app or simply use a web app. |
| **Development Tools** | Client side development will mainly be maintaining a website, so using VueJS framework would be an option. a Java IDE such as Eclipse can also be used. We will also require a website host, and a Java environment set for the server-side operations.  We can utilize Homebrew on macOS to aid in the web development process. To test functionality we can utilize tools such as VMWare Fusion to test the image interactions, and Selenium to test consistency.  Languages: HTLM, PHP, CSS, Java and JavaScript  SQL Database  Costs:  -VMWare: $199  Free: Homebrew, Eclipse, SQL, and VueJS | Similar to MacOS we can utilize a web-host such as WampServer, VueJS framework, and Eclipse as Java IDE.  Sauce Labs would be used for on-demand cross-browser testing.  Lots of distros are available for use both Debian and Fedora based. We can also use others such as Alpine and Flatcar for docker supports to help with image management.  Languages: HTLM, PHP, CSS, Java and JavaScript  SQL Database  Costs:  Free: Various Linux Distros, Eclipse, SQL | Similar to MacOS and Linux we can utilize a web-host such as WampServer, VueJS framework, and Eclipse as Java IDE. One additional option would be to use Microsoft's own web server Internet Information Services.  Given that command line tools are natively supported on unix based OS such as Linux and MacOS, we can use WSL (Windows Subsystem for Linux) to run a Linux-like file system, CM-line tools, and GUI apps.  Languages: HTLM, PHP, CSS, Java and JavaScript  SQL Database  Costs:  Free: WSL, Eclipse, SQL | Assuming that we will be utilizing native applications, one option would be to use a multiplatform tool such as Xamarin, which allows us to operate under both Android and iOS. This also provides easy integration with MySQL and is open-source.  We can utilize RAD (Rapid App Development) toolset which allows for both building and hosting through systems such as Buildfire. Can use Atom or Xcode with JavaScript.  Costs:  Free: Atom, Xamarin, Visual Studio, MySQL |

**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**: Given the requirements set forth, the most appropriate operating system would be Linux, allowing *Draw It or Lose It* to expand its current environment. One prominent proponent in this consideration would be the cost and the lack of expenses due to Linux being open-source and having no notable licensing fees. The advantages of Linux in comparison to other operating systems also include better security and scaling. Using this OS in conjunction with a cloud-based architecture allows offloading several concerns to established sources. Examples of these include Amazon S3, Oracle Cloud, and Microsoft Azure.
* **Operating Systems Architectures**: Linux operating systems are open-source and widely regarded as very secure. They are UNIX based and utilize the Unified and the Extended File Systems. Breaking down its configuration, the base begins with the hardware involved in the system. The next layer is the Kernel space which essentially takes commands from the Shell, as well as some commands from outer layer instruments such as applications, and interacts directly with the hardware. The next layer, the Shell, uses system calls to directly contact the Kernel space which provides fast access to hardware. The outermost layer contains the remaining components needed such as applications and compilers. Using this layer-based approach allows provides better efficiency, reduces disk space, and provides better communication between the user and the involved hardware. Utilizing both static and dynamic libraries also helps with faster processing and provides better security. Given that *Draw It or Lose It*’s expansion is the main goal, the Arch Linux distribution could be a viable option when used in unison with some pieces of development such as MySQL, PHP, and Apache.
* **Storage Management**: The key aspects that are required of a storage system include security, easy access for both developers as well as multiple unique clients, multi-platform access, and finally cost efficiency. Virtualization via cloud storage is an option when used in conjunction with direct-access storage solutions. The most commonly used cloud storage will be Amazon’s S3. Use of the xfs filesystem may also be helpful for storing and then accessing needed files. XFS also provides increased scalability for input/output threads and file system sizes. Linux is also able to use blocks to help with storage, and especially helpful for our case with recalling images for rendering by decreasing the time needed to locate the said file in the disk.
* **Memory Management**: With the help of Linux we can utilize the Virtual Memory Primer to help with memory management. By using demand paging we can securely share data across multiple processes. We also have the option of using Zones by allowing Linux to group memory pages based on their possible usage. For quick access, memory management uses page caches which avoids needless retrieval of data from central storage. Making use of efficient network headers of TCP/IP for calling images or other data during game operations. Optionally memory hotplugs can be used to reduce memory usage and to also implement or uninstall NUMA-nodes. Looking at the memory management process an operation is delivered to RAM which then creates a page cache of the requested data after obtaining it from storage. Any future processes requested will use the CPU to build a page cache.
* **Distributed Systems and Networks**: Linux offers us several tools regarding distributed systems and networks. We have the ability to isolate network resources by virtualizing them into containers, each with its own dedicated network stack. We also are able to use network drivers, specifically, the bridge, host, and overlay to help with outages and connectivity issues. There are three main types of distributed system architecture which include 3-tier, multi-tier, and peer-to-peer. The 3-tier approach involves the client sending a request via input to an intermediary middle tier, which is then processed and relayed to the server. For our purposes and due to the lack of significant complexity within the game and the web browser reliance, multi-tier and peer-to-peer architectures are unneeded options. This is mainly due to the overhead needed for both these systems as well as the cost of maintenance.

Focusing on client to server communications and increasing load capabilities due to increased demand from expansion into multiple platforms is key to create a well performing program. One of the biggest issues with reducing downtime is both the cost and the resources associated with it. By using load balancing we can hope to remove single points of failure thus allowing us to run as multiple nodes. Some load balancing algorithms include a round-robin connection to allow for low latency responses with concurrency. Other ways to maintain connectivity is by using REST APIs which allow of updating specific elements without complete shut down of the game. Server scheduling also helps cover possible server outages or DDoS attacks. Using third-party fault tolerance systems is also an option, which reduces the stress on the current resources (hardware and software), at the expense of financial cost.

* **Security**: The main duties of security will involve protecting game integrity and protecting user information. Linux inhertinly is very secure. It utilizes the Virtual File System where "everything is a file," which allows us to place restrictions as seen fit. Linux also uses packet filters in the kernel for an additional level of security. It is also fairly easy to enable HTTPS on our web server through Linux by generating a CSR. Linux also limits users privileges by utilizing DAC which polices who can access specific files. Virtual address mapping is also another tool used by Linux so any malicious system calls made by the user will be localized and only affect the logical address instead of the physical address. This is possible due to the separation of the user-space and the kernel-space. Other actions we can perform would be routine testing via Black box and Gray box testing. These help identify potential risks such as broken access control, SQL injections, misconfigurations of XXEs, and even sever side request forgeries. Next we can use a web application firewall which will allow us to filter incoming HTTP traffic by operating in the protocol layer. Runtime application self protection is also a tool which is available to us which can analyze user behavior in real-time to target sources of potential attackers, which then can terminate the identified session. Data breaches via vulnerabilities in APIs can be patched up by running vulnerability scans and simple pre-emptive actions such as checks to make sure all ports are closed when not in use. The core aspect of most security is user authorization. By using role based access control we are able to clearly define access rights to users and admins. Encrypted autheticantion may be possible via EtM (Encrypt-then-MAC), which provides extra security during verification.

Works Cited

emertxe\_admin. (2022, January 19). *Why is linux more secure than other operating systems?* Emertxe. Retrieved October 16, 2022, from https://www.emertxe.com/embedded/why-linux-is-more-secure-than-other-operating-systems/#:~:text=When%20a%20Linux%20system%20is,virus%20in%20systems%20with%20Linux.

Kent, S. (1970, December 1). *IP encapsulating security payload (ESP)*. RFC Editor. Retrieved October 16, 2022, from https://www.rfc-editor.org/rfc/rfc4303#section-3.3.2.1

Krawczyk, H. (n.d.). *The Order of Encryption and Authentication for Protecting Communications (Or: How Secure is SSL?)*. Retrieved October 16, 2022, from https://www.iacr.org/archive/crypto2001/21390309.pdf

Osmani, A. (2016, September 3). *Offline storage for Progressive Web Apps*. Medium. Retrieved October 16, 2022, from https://medium.com/dev-channel/offline-storage-for-progressive-web-apps-70d52695513c

*What is a distributed system? - an introductory guide*. Confluent. (n.d.). Retrieved October 16, 2022, from https://www.confluent.io/learn/distributed-systems/

*What is fault tolerance? definition & faqs*. Avi Networks. (2022, March 18). Retrieved October 16, 2022, from https://avinetworks.com/glossary/fault-tolerance/#:~:text=The%20goal%20of%20fault%20tolerant,mission%2Dcritical%20applications%20or%20systems.