

Gameroom.com

**CS 230 Project Software Design Template**

Version 1.0

**Table of Contents**

**Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | 03/22/2024 | Chris Gauthier | Version 1.0 - Initial Release |
| 1.2 | 4/6/2024 | Chris Gauthier | Version 1.1 - Document Update |
| 1.3 | 4/20/2024 | Chris Gauthier | Version 1.3 - Document Update and Finalized |

**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**

Creative Technology Solutions (CTS) has been contracted to provide a solution to developing and integrating the current Android based game **Draw It or Lose It** into a web-based platform accessible from multiple platforms. Developers of the current program do not have the expertise in developing the program for a multi-platform environment.

Draw It or Lose It is a game where teams compete to guess an image that is rendered over the course of a minute, with four rounds per game. Each game can have multiple teams with multiple players on each team. Team and player names must be unique, and only one instance of the game can be in memory at a time.

**Requirements**

* Ability to have one or more teams
* Ability to have multiple players on a team
* Game and Team names must be unique
* Only one instance of the game can be loaded into memory
* Cross-platform compatibility (web, iOS, Android)

**Design Constraints**

Development for web and iOS will have different requirements for completion. iOS version will require acceptance into the App Store. Web based version will need to be hosted through a website, custom launcher, or gaming hosting platform. The program must integrate to each platform to ensure game rules and design are synced and stored, such as game, team, and player names and Id’s in addition to the timer and image library. Program design will require a Singleton design pattern to ensure only one instance is created in memory.

**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 Class uses direct association with the Singleton Tester Class to ensure there is only one instance of the Game Service running at any one time. The Entity Class is the parent class to Game, Team, and Player classes; thus the child classes inherit attributes from the Entity Class. The GameService Class has Association with the Game class where GameService can have many objects of the Game Class. The Game Class has Association with the Team class, where the Game Class can have many objects of the Team class. The Team Class has Association with the Player Class, where the Team class can have multiple objects of the Player class. Lists are created to maintain lists of Game, Team, and Players objects where each new addition of game/team/player is checked against the list to ensure the new addition is unique.



**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:**  \* Likely programed in Swift.  \* Less popular than other OS's  **Advantages:** \* Seamless integration within the MacOS ecosystem.  \*Native support of other languages including Python and Ruby.  **Weakness:**  \*Typically higher cost to operate.  \*As of 2022, MacOS Server has been discontinued. | **Characteristics:**  \* Most commonly used server OS.  \* Lean design and highly customizable  \* Free to use (No licensing cost)  **Advantages:**  \* Open source which allows access to a vast repository of resources  \* Highly stable  \* Easy integration with web services and other platforms.  \*Huge pool of engineers  **Weakness:**  Certain programs or applications may not be available in Linux. | **Characteristics:**  \* As common as Linux for use as Server OS.  \* Requires Microsoft licensing    **Advantages:**  \* Compatible with all Microsoft products and services. Including Azure for cloud service.  \* Most widely used enterprise OS for stability and support.  **Weakness:**  \*May have issues utilizing open source software.  \*Licensing cost could be a factor if budget is tight. | **Characteristics:**  \*Not designed for server application due to compute power, integration, and physical hardware.  **Advantages:**  \*Only for localized use.  **Weakness:**  \* Limited compute and memory power.  \* Lack of integration with other OS's  \*Not designed for scalable hosting |
| **Client Side** | **Software Development Considerations**  **Cost:**  \*Mac devices are more expensive in general. Mac book pro costs $1600-$2000/device.  \*MacOS dev software can be free, but may be limited by functions and features.  Apple Dev Program license is %99/user  **Time:**  **\*** Mac hardware in general typically performs faster than Windows OS's, in addition to better OS updates, which cut down time. For large scale projects or enterprise applications Mac devices can potentially save hours over the course of a year.  **Expertise:**  MacOS is a common enough platform that finding developers should be too difficult, however, MacOS dev skills, may not transfer as easily as other platforms. | **Software Development Considerations**  **Cost:**  \*Cost can vary however there are very few limitations on what can be used. Off the shelf laptops can run from $400-$1600 depending on hardware. Any WindowsOS device can be re-imaged to run Linux  \* Linux Software is free and open-source  **Time:**  **\*** For developers not familiar with Linux, it can take time to train, however most developers will have a baseline understanding of Linux applications.  \* Large repository of open source material could cut down on development time.  **Expertise:**  \*Developers will need to be able to leverage open source material and understand integration into the application.  \*Must have understanding of Linux Command Line tools and utilities. | **Software Development Considerations**  **Cost:**  Same as Linux, with the addition of Windows OS license.  **Time:**  \* Most common development platform, therefore most developers will have experience in the environment which will allow for faster development.  \* At enterprise scale, small slowdowns to development such as OS updates, and slower hardware, could effect/cost resources.    **Expertise:**  \* Need proficiency in .NET framework and Visual Studios. | **Software Development Considerations**  **Cost:**  \* Potentially double the costs due to the need for two devices to develop on (iOS and Android).  \* Software for development is free use.  **Time:**  Developing on iOS and Android, requires multiple languages and more time spent ensuring integration into each platform/ecosystem.  **Expertise:**  \* Needs proficiency in two development environments and be familiar with both platform requirements(Apple Store and Google Store). |
| **Development Tools** | **Programing Languages:**  Swift  Objective-C  C/C++  Ruby  Python  JavaScript  **Tools (IDEs and Deploying):**  IDE: XCode (Official macOS)  Deployment through XCode | **Programing Languages:**  C/C++  Ruby  Python  JavaScript  Java  PHP  **Tools (IDEs and Deploying):**  IDE: Visual Studios, Eclipse, IntelliJ  Multiple deployment options | **Programing Languages:**  C#  C/C++  Visual Basic .NET  Java  JavaScript  Python  **Tools (IDEs and Deploying):**  IDE: Visual Studios (Official for Windows)  Eclipse  Multiple options, Microsoft specific: Visual Studios and Azure | **Programing Languages:**  Swift and Objective-C(iOS)  Java and Kotlin (Android)  **Tools (IDEs and Deploying):**  IDE: XCode (iOS)  Android Studios (Android)  Deployment through respective app stores. |

**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**: It would be recommended to use a Linux based platform for server-side operating platform with a serverless architecture. The suite of Amazon Web Services (AWS) products will be the perfect balance of cost, reliability, scalability, and familiarity.

Client-side will be platform specific (PC - Windows, MacOS, Android, iOS) and because communications between server and clients will be through RestfulAPI’s, programing language can be specific to platform. Additionally, if expansion to other platforms is desired, such as gaming consoles, no changes will need to be made to the server-side and being hosted in a robust cloud environment allows for seamless scalability.

* **Operating Systems Architectures**: The required hardware for a Linux based cloud environment can be kept to a standard “off the shelf” service such as Amazons EC2 Auto Scaling package. The server side of the game application will only need to compute, security validation, game logic, and game session data as well as store a small (relative) library of images (approximately 1.2GB). Since the client-side application will be handling the rendering of the game, data packet transfer and connection will be less impactful to the end user.

For the client-side, Operating System architecture will be platform specific. Specifically for the desktop version, two options are available, either an executable game available to download from a host (Steam, Epic, or directly from the developer), or the game could be browser based through any web browser. Mobile platforms, both iOS and Android should be delivered via the respective app stores.

* **Storage Management**: Server-side storage management will be maintained on AWS server hardware. Due to the relative simplicity of the Game Room application, physical storage requirements will be low. For client side, since a large portion of the game (relative to size), the image library, will be stored on the server, the overall application size will be kept to a minimum, allowing for even the most limited devices (mobile platforms) to store the required application on in their internal storage. Storage management will be kept to a file system vs database structure due to the limited amount of I/O and executions that will happen in any given game session.
* **Memory Management**: Overall memory management should not be a major concern with the Game Room application, as the demand for memory will be limited. Specifically, on the server-side, Linux employes a Virtual Memory System, which swaps active and inactive files from physical memory on demand. This allows the program to “use” more memory than is available. On the client-side program design will ensure that memory requirements are kept to a minimum by only requiring the immediate need of the game on the clients system. This will be accomplished by the client-side only loading a single active image. Once that game round is over, the client will get the next image from the server and load it into memory for the next round.
* **Distributed Systems and Networks**: Distribution and connectivity will be straightforward for a few reasons. First, all game logic, libraries, and user data will be handled on the server side, with clients “connecting” to the host server and communicating through HTTP request. This ensures that no matter the device, the user can connect to the game services and their profile will be maintained. Furthermore, by having a server based host, connectivity through a game session will be more stable, since it will be only relying on the clients connection to the host, vs a peer to peer, where each client is connecting to each other.
* Reliability is achieved by the use of the cloud host (AWS) by way of redundant services, 24/7 service monitoring, and regional data centers. A robust network ensures that if a single piece of hardware malfunctions, the system is designed to seamlessly transfer data to another location.
* **Security**: With a server-side authentication application and the principal of least privilege applied the Game Room application, integrity will be maintained. With security in mind, developers will create “role” based assignments for clients ensuring each client has just enough privilege to execute the game functions but not enough to modify any game files. Additionally, all client-server communications will be utilizing SSL protocol ensuring transmission between client and server is encrypted to avoid intrusion. Finally, the firewall protection provided by the server hosts (AWS) will provide additional security to direct attacks on the server application and unauthorized access to game files.