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 | 08/14/21 | Franklin Lewis | Update recommendations, complete section for memory/storage, operating platforms/architectures. |

**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**

Draw It or Lose It requires a game environment that will allow a consistent web-based play experience across multiple client devices. Further the game should only allow a single instance of each game be held in memory at any given time. Each game shall allow one or more unique teams comprised of one or more unique players to compete against each other. A Single instance of a Game Service will be implemented to handle the creation of new games, teams and players. This ensures that each game will only exist a single time in memory, the teams per game will have unique names, and the players of each team will have unique names also. A consideration should be made as to whether player names should be unique per team, or per game to prevent confusion of players with the same name, on different teams.

**Design Constraints**

A web-based implementation of this game has some ambiguity attached to it, if the game is to be played via browser on multiple platforms, this has fewer constraints; due to commonalities of browser support across various devices. In this solution, a player is able to create, find, or join a not yet started game via their browser, where the server side will manage the game flow and communications between the clients and their browser. If this project is to run as native applications or simply java applications on multiple devices as a cross platform solution... differences in platforms may lead to subtle visual differences, as well as render time for the portions of the images to be drawn to the client. Another consideration for the design of how a guess is submitted is needed as this too could lead to some platforms sending their guess slower or faster. These would lead to some unfair advantage on one platform or the other. To solve this issue a simple system can be implemented to include the current client state with their submission, compare to the expected state, and adjust for any delay to determine the true fair order of each submission.

**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 below outlines a Singleton pattern for the GameService class, this allows for a single instance of an object to track, maintain and update unique game instances per the game model. This class has zero or more game objects, team objects, or player objects. Although these three objects are their own type, they inherit from a base class “Entity”, which allows for non-repetitive code, as well as tidy code when handling these different objects. Common methods across any one type of Entity (Game/Team/Player), is defined within the Entity class, allowing any of the child classes to use these within their own class (per object instance). The class specific methods for any one type of Entity (Game/Team/Player), is defined within the actual child object class. These methods are specific to each Object type and unique from methods within the other Object classes. The program driver, holds the main method, which instantiates the game service, and allows for games to begin being create, as well as adding teams and players to those game instances. The SingletonTester class, tests and ensures that the game service class, is only existing as a single object in memory as well as if it is referenced from another class, it is given a reference to the original object.

**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** | Due to Mac being able to host java applets this is possible and has few constraints, however there is added overhead due to GUI. Furthermore MacOS requires purchase of a license to use it. | Linux is well versed in hosting situations, allows for minimal overhead, and optimal performance. Linux requires no licensing and can be used for free. A Linux cloud server would suit this project perfectly, as it is low cost and scalable on the fly. | Although windows is able to act as a host, the overhead from the windows services and GUI would not be ideal. Apart from this hosting a windows server requires stronger hardware for the same performance, expensive licensing fees come along with windows server as well. | Using a mobile device as a server-side host would prove to be difficult due to the nature of addressing over cellular networks as well as performance. In theory hosting network could be established however the stability and ease of doing so would greatly outweigh any benefit. each mobile device and provider would introduce further conflict. |
| **Client Side** | MacOS by default comes pre-installed with safari as it’s primary browser, our project will utilizing the REST API will rely heavily on the JavaScript implementation. Safari uses JavaScriptCore/WebKit... However, >80% of end-users install and use Chrome which utilizes the V8 engine. | The majority of Linux users, utilize chrome, firefox, or a derivative of one of them. This again leads us to V8, and in the case of FireFox, SpiderMonkey. Due to the client side being HTML/JavaScript, Some visual appearances may have slight variance. | The most popular browser Utilized on windows is Chrome, or a chrome derivative as well as FireFox. Just as with linux these browsers are the most prominently used. Some variations exist between SpiderMonkey and V8, however they are mostly backward compatible. | For Mobile Devices, There is a majority of users still utilizing Safari on Apple devices, and Chrome taking up the remaining portion with a small margin between the two. Since Mobile browsers are easily detectable, it is trivial to display designs specific to them. Care will need taken to test and ensure API calls function properly. |
| **Development Tools** | For this project Eclipse is available for all platforms, to cover the server side of things. When Developing the Client-Side of things, many programmers will use a free tool also available on all platforms called Visual Studio Code. Personally I develop HTML/JS in a standard text editor with code highlighting such as Sublime Text. | Again the same development tools apply for linux as does mac. However many programmers (such as myself) when developing on Linux, tend to use Vim or another available text editor such as Joe’s Editor, or even one they made themselves. This also applies to Java programming quite frequently | On windows, Java programming is frequently done using either Eclipse or NetBeans. Personally I prefer NetBeans... For HTML/JS My personal favorite is Notepad++ or Programmers Notepad. These tools have effective syntax highlighting and completion as well as the ability to quickly run tests with configured commands. | Due to the projects nature... little development will take place on a mobile device. Rather be accessed and used through the browser. However it can be done on a mobile device with a text editor... however would prove cumbersome. If needing native apps that impliment a common JS engine then iOS/Android specific tools would be needed to create these applications. |

**Server, Client, Development Conclusions**

My best assessment of this project, based on years of experience and research to the specific requirements of this project, leads me to three selections. Linux for the server platform, primary client target should be chrome, and development will take place on our current configuration of free tools, Eclipse, and Notepad++.

The server platform selection requires the ability to maintain costs at a minimum while being able to optimize performance and scalability. With the lightweight nature of Linux servers, overhead is kept to a minimum and higher performance is achieved from equivalent hardware. The free price tag attached to Linux immediately saves costs that can be directed toward scalability of larger or additional servers to balance the load.

The choice to primarily target chrome is determined by it’s current and past performance as being the leading choice for consumers to use as their standard browser. Proper testing should be completed for visuals and functionality on Safari (to ensure the majority of mobile users have the same expected experience), as well as Firefox to ensure that 99% of desktop users, as well as 99% of mobile users are accounted for in the development phase. That being said... rigorous testing for appearance and functionality on mobile devices will also be needed to ensure the devices are served the content best suited for their browser.

These conclusions are based on best practices, years of experience with developing for multiple browser deployments and are the optimal selections for player experience and company cost/expansion.

**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**: The server side, should be handled by a Linux system to minimize overhead and cost written with a RESTful model. The client platform should be web-based via HTML/JavaScript, targeted for the most popular browsers, as well as mobile browsers.
* **Operating Systems Architectures**: Linux for the server side allows for better security, ease of development, and compatibility as well as maintainability. The costs for linux servers is lower, the overhead is less, and scalability is smoother. The client decision of Browser allows for development and compatibility for the largest variety of mobile devices, of differing performance.
* **Storage Management**: Storage management recommendation for the server side includes web-optimized images, to get the optimal capacity requirements. PNG files can be stored as BASE64 encoded strings in a database, for easy management and scaling. The client side should use browser storage for up to 3 rounds in total (current + next 2), this ensures players are not effected by differences in loading times.
* **Memory Management**: Games should only exist in memory while they are active, and not complete. A complete game, shall be saved to the database, and the game instance destroyed. Inactive games should exist for a determined amount of time, due to the nature of the game a game should expire after this time and self-complete or be destroyed. Server Side should only require hash pairs, to know what an image is for any individual game, rather than keeping the images in memory. The client side, will discard used information as the next round begins. A cache system on the server side will allow more consistent memory utilization as data is being transferred to clients.
* **Distributed Systems and Networks**: Due to multiple client platforms, the connection method to the server-side, should be handled over HTTPS via AJAX calls to the REST API on the server side, to update clients with the current state of the game as it progresses. A game should be created, the teams are established, the players are created, finally the game will begin. The state of the game is communicated with the clients, per-update or action by themselves or another player. Should a player lose connection due to server or client outage, games should exist for a pre-determined time, and continue as long as competing teams have a functional number of players remaining. Gameplay should pause for a pre-determined amount of time on detection of connection loss, to allow all players this time to re-connect prior to the game resuming/ending.
* **Security**: To protect player data as well as server information, connections to the server and client will be made via SSL on HTTPS protocol. Clients have no direct path to other clients, rather each client communicates to the server game service, this game service updates the game state and updates the clients individually with this information. The smallest amount of game information required for successful gameplay is transmitted/received. Server expected states (current, past, and possible), are used to compare client transmitted information, rather than blindly accepting and processing client submissions.