

<Name-of-Software-Application>

**CS 230 Project Software Design Template**

Version 1.0

**Table of Contents**

**Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | 03/22/2023 | Eladio Yanez | I went over the executive summary, requirements, the design constraints, domain model, evaluation table, and recommendations for the project. |
| 2.0 | 03/29/2023 | Eladio Yanez | I checked over the evaluations table again and adjusted it with my knowledge now. |
| 3.0 | 4/13/2023 | Eladio Yanez | I went through and properly adjusted the recommendations for the project with the knowledge I have now on the topic. |

**Executive Summary**

Our client does not know how to set up the environment necessary to make their "Draw It or Lose It" game work with the general ruleset they want to work on a web-based version of the game. It is our job to ensure that their project can work on web platforms so the game works.

**Requirements**

*\* The game must have one or more teams with multiple players on said teams.*

*\* The player names and teams must be unique and can only be used once when choosing a team.*

*\* Only one instance of the game can exist.*

**Design Constraints**

\* Only one instance of the game can exist I think is the major design constraint of the problem.

**Domain Model**

\* Program driver utilizes singletonTester to see if its operable.

\* Game services have a list of games, as well as private variables for the next game, player, and team id as well as the private constructor for GameService. It utilizes addGame, getGame, and the rest of the public methods to get the resources the parts reliant on it use.

\* Game, Team, and Player all extend from Entity which means they inherit from Entity.

\* Game is associated with GameService because it utilizes information from gameService, while Team is associated with Game because it utilizes information from Game, and Player has an associate with team. The amount that each part of the code uses can be infinite if the number of games kept going, they do not stop at a set instance.

\* Game registers information regarding team and adds it to the team, thus its association, team does so with player, and gameService does so with game, which again explains the general association. (addGame/addTeam/addPlayer.)

\* Entity utilizes accessors to get general information.



**Evaluation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| **Server Side** | It is easy to access and configure the server and to work on the UI. Because Mac is a closed platform, it does not have to deal with loading times as much as other software competition. | The platform is the most difficult to navigate, has access to command shell, but rough. | The most expensive out of the bunch, but the experience is extremely user friendly, and it has command prompt. | Due to the fact mobile devices are broad and it depends on the model of the phone, brand, and other factors, this is hard to judge. This does hurt it compared to the other OSes though. |
| **Client Side** | It is extremely expensive to make, navigation is difficult if not specialized in it in the first place. | This requires natural understanding of Linux to make the most of it, the least intuitive to use compared to the other OSes. | The friendliest OS for a casual consumer, though it is worth noting it is more expensive. The lack of a skill barrier is preferrable for the client though. | Flexible, but it requires a lot more effort to implement compared to other devices. |
| **Development Tools** | Most standard development tools that you would get on other operating systems are available on mac, though it is slightly more limited than the other OSes. | It has many of the most important tools other OSes have, but due to the fact using Linux can be more complicated for the average person compared to other OSes, it might not be worth it. | Due to being the easiest to use, while also having access to many of the IDEs that other OSes can use, the most favorable out of the bunch, I think. | The most limited to work for compared to the main operating systems. |

**Recommendations**

* **Operating Platform**: Windows is the best out of the bunch, the most user friendly compared to the other OSes. It helps Windows has so many IDEs it can use as well
* **Operating Systems Architectures:** Windows' operating architecture is user-friendly. By splitting everything into the User and Kernel modes, it allows for a strong ease of use for those unfamiliar with Windows while those with the knowledge of the system to manage everything they need to manage that is obfuscated in User Mode. It is a lot easier to use with this system than otherwise.
* **Storage Management:** Microsoft Azure (Window’s storage system) is the best we can get within this context, it is user friendly, it is cloud so we can continually increase the amount of data we need with continual payments which makes expansion for the job easier and easier, and it is efficient more than anything else. Overall, it is a strong option worth opting into.
* **Memory Management**: Window's memory management is strong and transparent, you get a good understanding of the amount of ram the program is using, you will be given the vital information you will need. It is a solid foundation.
* **Distributed Systems and Networks**: Distributed Systems provide an immensely vital role in the success of Draw It or Lose It, the distribution of information between the Clients and the server is the primary focus due to the fact in enables the game and ensures a stable flow. In that regard, applying the REST method would help ensure that the day data is retrieved, stored, and managed will ensure a smooth experience both on a systematic level for server, and on a user level by providing a smooth player experience that keeps information transparent and professionally managed. Furthermore, it makes management easier, so it makes solving issues within the system far more manageable. Distributed systems especially help avoid redundances within the game system and allow the game to avoid wasting the player’s time by redirecting the server.
* **Security**: Windows security is admittedly the worst out of the bunch and the other OSes can handle it better, but Windows is serviceable enough in this context I believe.