<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	6/2/2024	Alexis Mims	Filled in Executive Summary, Requirements, Design Constraints, and Recommendations
2.0	6/16/2024	Alexis Mims	Added to Executive Summary, Requirements, Design Constraints and Recommendations. Updated system architecture view, evaluation, and recommendations.
3.0	6/23/2023	Alexis Mims	Update information of the architecture, design constraints, and recommendations, also organized the evaluation of each platform

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 objective is to design a web version of an android based application called draw it or lose it. This game aims to allow multiple teams to be able to play at the same time, through multiple platforms simultaneously by only allowing the game to be played with one instance.

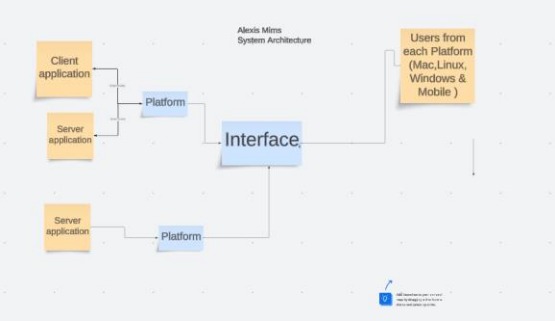
Requirements

A web design that is compatible and functional with multiple teams and players without collision. Different identities are needed for the game and team as well as the ability to check if the game is already in use.

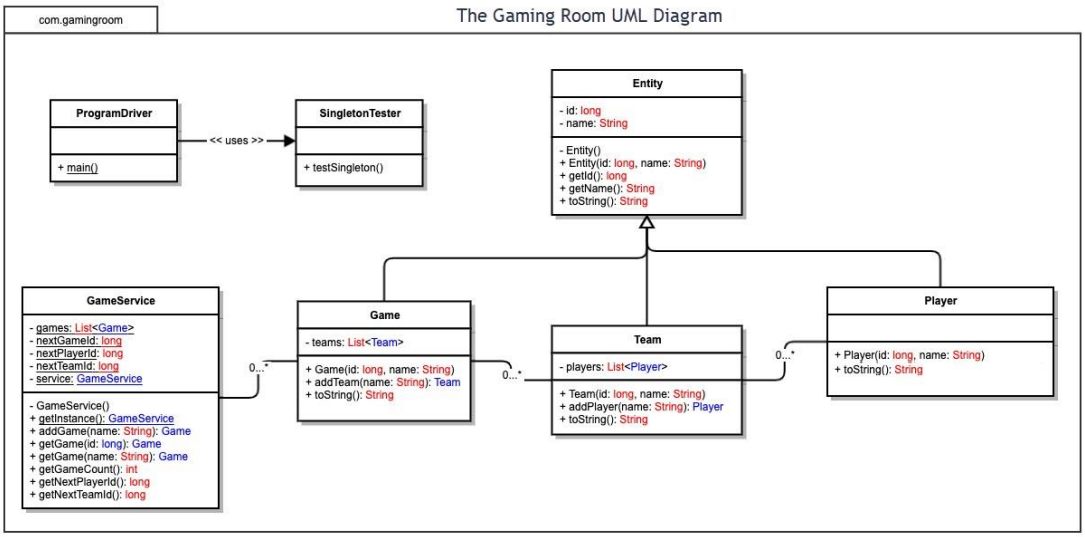
Design Constraints

The constraints within this design include being compatible with both mobile(android/iOS) and web development which is why java is a perfect language choice since it can be used for both. It also must be able to allow teams from different platforms simultaneously. Other constraints include the applications needs to be adapted to multiple programing languages to be compatible with all platforms. The applications needs to way to be able to authenticate information, and also efficient storage and memory allocations to prevent further issues.

System Architecture View



Domain Model



The design shows a main driver class that starts the creation of the game, through the GameService class which follows the singleton pattern. 'gameService' is then instantiated through the private constructors of 'get Instance()' method. Following that the 'addGame', 'addTeam, and 'addPlayer' methods use the iterator pattern to stop and prevent any duplicates and add new objects such as another game, another team, or to add on players. Then 'Game', 'Team', and 'Players' inherits from the 'Entity' class. This class has protected attributes 'id' and 'name' as well as default constructors that stop and prevent null objects. These are all safety nets that ensure the code follows the rules of object-oriented programming.

Evaluation

Development Requirements	**Mac**	**Linux**	**Windows**	**Mobile Devices**
Server Side	Mac only allows you to operate on their devices which in turn is a hefty investment, but overtime has not made major changes outside of routine software updates.	On the server-side Linux is great for web-based applications. It also has a lower barrier of entry in terms of cost. Linux also has better security when compared to MacOS and Windows.	Windows has expensive licensure but is also easy to set up and does not need a lot of maintenance but is more vulnerable to malware and spyware.	Allows you to tailor it to individuals in terms of accessibility and barrier of entry. Some drawbacks are it lacks power since most applications are cloud based and tend to be more vulnerable. Also not able to use multi-user serving and are not as scalable.

Client Side	User also must have a Mac Book or another apple device which is generally expensive in turn results in a lot of upfront cost, because it is not accessible out of Apple OS, and accounts for less than 20% of the market which leads to smaller opportunities, but Mac is user friendly in terms of learning the platform	Some pros of Linux include the affordability, it is an open-source OS, and easier to maintain. Linux has another pro of widely available support for users on its main platform. Some cons are. Coding on Linux would take longer here since they utilize languages such as C-programming and Python. Swift is over 2 times as fast.	It is highly secure and is compatible with most hardware. Also, they have advanced security, it also makes up about 75% of OS users, leading to larger opportunities. Some cons are that you must be versed in windows and pay for add on features which can contribute to upfront cost and cost over time.	Would need developers with extensive knowledge is app development and how to transfer the same interaction from web design. There is also a wide range of hardware choices
Development Tools	A MacOS is usually coded with the Xcode IDE. This could be coded in objective C, but mostly swift which is the main tool used by most Apple developers	Python is already installed on Linux devices so next would be an IDE such as visual code studio. Languages such as C/C++, and Java can also be used within Linux development, eclipse is a common IDE these can be coded in.	You have free choice of hardware needed, Visual code is also a good IDE because of the plugins, integration with the ability to efficiently edit code.	With android, android studio is needed and will most likely be coded in Java. For iPhone swift or C and an Apple device within the Xcode IDE. There are many IDEs' and languages that can be used to code and support a mobile app development some other popular option in include Eclipse and VScode.

Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

1. **Operating Platform**: The first choice for the operating platform would be Windows since it has a large database and makes up for the majority of OS Users. This in turn makes it more appealing to the developers not to mention the lower upfront cost. Also, Linux is another viable option because of the low barrier to entry and cheap licensure, also security and overall efficiency.

2. **Operating Systems Architectures**
   For the Backend Design Deploy a contemporary backend with microservices using containers in Kubernetes or Docker for the growth of the application. This is a function of the specific cloud provider as they have their own specific proprietary tools. For the Frontend Rendering to improve the server load and the continuous costs, let the frontend be responsible for the rendering. Client-side rendering helps in maintaining fluidity in gameplay, in the presence of network latencies; it also preloads a few images. In terms of the platform choice choose if the game is going to be online, located in a web browser, or if it is going to be a downloadable program for PCs or Macs. For example, a game that is played on a browser and implemented with the help of the Progressive Web App (PWA) technology might be easier and more flexible. Overall, the focus should be made upon the scalable backend services, the client-side rendering, and the browser game as it will be easier to deploy and to play.

3. **Storage Management**: For storage management SSD storage seems to be the best option because of the fast connection to assets. Then a Kubernetes node for file storage and a NoSQL node for managing the game data.

   Windows provides many solutions for storage and memory. An example would be Azure. This operating system supports memory for physical and virtual addresses. It is also well known. tools such Visual Studio and OneDrive which in turn allows optimized run times and efficiency. storage.

   Linux is also a good choice because it has concrete style system options, user support, and a lot of flexibility, it also supports a variety of file systems allows it to be more reliable along with being compatible with many storage solutions.

4. **Memory Management**: A system load monitor should be deployed to adjust the memory storage which in turn will allow efficient performance and limit cost.
5. **Distributed Systems and Networks:** Kubernetes with a cloud based set up will prevent failures during maintenance allowing a trustworthy platform. Azure can also be used with this network as it allows maximum uptime when using the cloud-based services allowing the client to keep up with feedback from their user and overall making a more manageable system.
6. **Security**: A role-based system would be implemented protecting the data from unwanted entities and only allowing the user to see the right information. Aura would be a great choice because it allows protection for Mac, Linux, Windows, Ios, and other mobile devices. It also allows customers to be able to reach customer support 24/7, which builds trust with the users of the application. Along with Aura, Azure is great because it allows only limited access per user, has option for VPNs when needed, has database security, and encryption options as well.