

<Draw It or Loose It>

**CS 230 Project Software Design Template**

Version 1.0

**Table of Contents**

**Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | <06/09/2024> | <Cody Adams> | <Brief description of changes in this revision> |

**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”, is an application that Gaming Room plans to create. It is a web based gaming application, which has the goal of creating a distributed gaming experience. We aim to make a scalable, maintainable, and efficient application that can support multiple players, and teams. This application will leverage object oriented principles and streamline a finished product that has clean and robust code.

**Requirements**

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

**Design Constraints**

There are some design constraints that are worth mentioning. The first is scalability. The application must be able to handle an increasing amount of users without losing performance. One way of overcoming this would be to implement efficient data management, and distributed processing within the application.

One of the next constraints would be maintainability. Maintainability can be overcome by making the code modular. By making it modular we can ensure future updates and bug fixes will go smoothly.

Some other constraints that we might have are for security, compatibility, and performance. On the security front the application must be protected from unauthorized access. We can do this by implementing security measures such as passwords to access the platform. Compatibility is another constraint. The application must be able to work on different operating systems, and devices. This will allow for a wider audience, and an overall better customer experience. Lastly we have a performance. The application must provide real time responses and ensure a smooth gaming experience.

In conclusion these constraints require the need to carefully choose technologies, robust architecture, and frequent testing to make sure the system meets the requirements.

**System Architecture View**

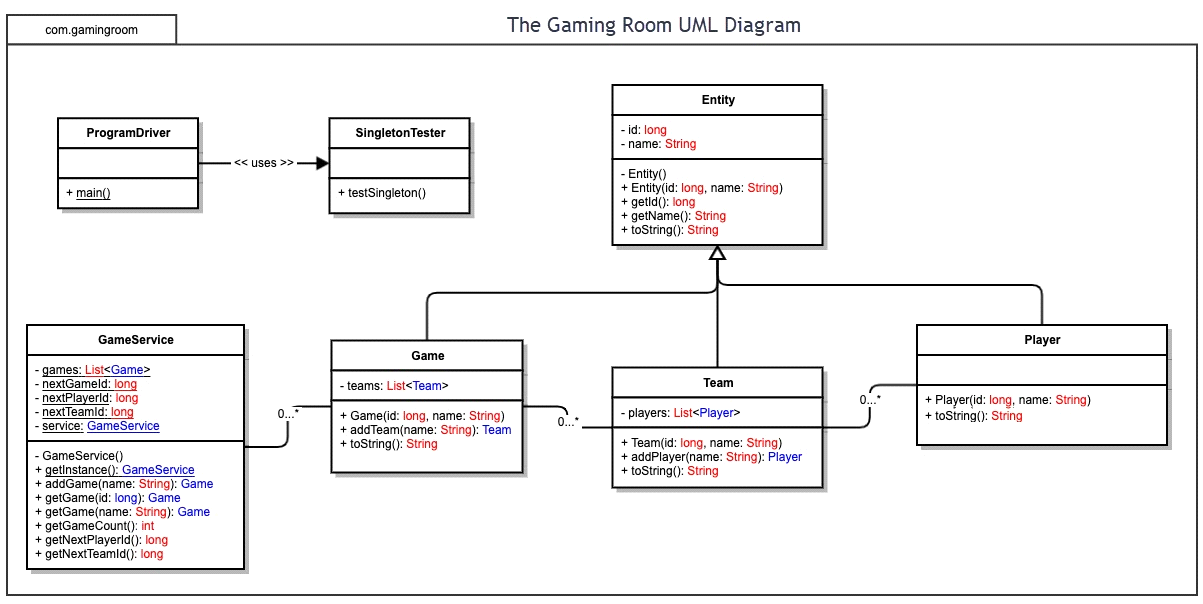
Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

**Domain Model**

* Entity Class: A base class including common methods (getId(), getName(), and toString()) and attributes (id, name).
* Player Class: Derives from Entity and represents particular attributes of players.
* Team Class: Contains a list of Player objects and inherits from Entity. It offers ways to bolster a squad with players.
* Game Class: Contains a list of Team objects and is inherited from Entity. It offers ways to include teams in a match.
* The GameService Class is responsible for managing a list of Game objects, creating new games, and tracking player, team, and game-specific identifiers.
* SingletonTester and ProgramDriver Classes: The primary method for executing the application is contained in the ProgramDriver class, while the SingletonTester class is used to test the singleton pattern implementation.

Principles of Object-Oriented Programming:

* Inheritance: To encourage code reuse and hierarchy, the Entity class acts as a base class for the Player, Team, and Game classes.
* Encapsulation: To safeguard the integrity of the data, every class encapsulates its data and offers methods to communicate with it.
* The GameService class uses the singleton pattern to make sure there is only one instance of the service, giving users a single point of entry to game administration.



**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** | Can serve as a server. Its weekness is it is not cost effective. It has restrictive licensing. It is strong in small opperations, that require Apple software integrations. | Linux is known for its well rounded server capabilities. Its stable, secure, and scalable. Another strength is that most distributions are free and cost effective. | Windows is used commonly in business and academic enviornments. They provide support for their .NET frameworks and other Windows tech. The downside here is higher licensing costs. | Most of the time these devices are not used as servers. They can still handle server like tasks. Examples of this are with data processing, storage, and cloud based backends. |
| **Client Side** | Apple is expensive when it comes to the computers themselves. The development tools are usually free. Moving apps through the app store will also involve a membership fee. On the upside Mac could help save time due to the streamline nature of their development tools. As for expertise, knowledge in Objective-C, and Swift would be recommended. You will also need to know Apples design guidelines. | Costs are low with Linux. Infact the costs are usually free. There can be costs for services outside Linux that build off of it. Think support services, or specialized distributions. There is a large variety of Linux distributions. This could increase our development time because of testing and making sure there is compatability accross versions. Developers should have knowledge of the different Linux distributions, and also using open source tools. | For small teams Windows could be cost effective. The use of Visual Studio could save money. For larger teams, the licensing fees could add up. The support Windows gives is great, and the access to tools like Visual Studio offers good support. This support can help decrease development times, and help with solving problems faster. Developers here need to be knowledgable with .NET, C# and ASP.NET for web applications. | Mobile devices will very. On the Android side cost can be low due to the tools being free for the most part. As for IOS development, this requires a Mac. These can be expensive. Another thing to consider is if we want the application to be on both platforms, this will require potentially double the testing, and maitenance. This could increase development time. For this developers need to be knowledgable in Java, Kotlin on Android, and Objectice-C, and Swift on Apple. |
| **Development Tools** | Swift, Objective-C will be the main ones. You may also use HTML, CSS, JavaScript, Python, and Ruby. Your main IDE will be Xcode. | Python, Java, C/C++, and JavaScript.  Common IDEs: Eclipse, Visual Studio Code, IntelliJ IDEA. | C#,.NET, ASP.NET  Visual Studio will be the main IDE. You may use PowerShell and other scripting languages. | We would use Java and Kotlin for Android, and Swift and Objective-C for IOS. Here we can use cross platform tools such as Flutter, React Native, and Xamarin.  For IDEs: Android Studio for Android, and Xcode for IOS |

**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**:

For the operating platform it would be recommended to go with AWS(Amazon Web Services). This platform will meet our requirements of being scalable, and allow for the growth of the “Draw it or Lose It” game app. AWS will let us carry this app across multiple computing environments. The benefits of AWS really comes into play with its services, and tools. The platform is secure, its high performing, and ultimately gives us the tools we need to create the gaming app.

* **Operating Systems Architectures**: <Describe the details of the chosen operating platform architectures.>

With the AWS platform we can run operating systems such as Linux, Ubuntu, Amazon Linux, and Windows. We can do this through virtual servers, and this will give us flexibility to know our app can be tested and run on multiple platforms.

Through this we will have auto scaling, which will adjust to the number of virtual server instances in response to traffic patterns. We can see what kind of loads the application can handle efficiently. We will also have Elastic Load Balancing, to distribute incoming application traffic, across our virtual server instances.

* **Storage Management**: <Identify an appropriate storage management system to be used with the recommended operating platform.>

For storage management we have Amazon S3. This will provide us with scalable object storage, which is great for organizing and storing game assets, backups, and user data. This is all while being very secure, and usable. We can also take advantage of Amazon RDS. This gives us access to use managed relational databases. This will allow us to utilize MySQL, which is great for storing user data, game data, and other related information.

Another would be Amazon EBS, which is great for block storage for our server instances.

* **Memory Management**: <Explain how the recommended operating platform uses memory management techniques for the Draw It or Lose It software.>

With AWS we can take advantage of elastic load balancing and auto scaling to manage memory effectively.

* **Distributed Systems and Networks**: <Knowing that the client would like Draw It or Lose It to communicate between various platforms, explain how this may be accomplished with distributed software and the network that connects the devices. Consider the dependencies between the components within the distributed systems and networks (connectivity, outages, and so on).>

With AWS we can use multiple computers to work together as a single system. This allows us to be scalable. Specifically this adds nodes to dynamically handle workload growth. We can achieve availability, by fault tolerance, even if an individuals computer fails. This ties into consistency because since we have multiple computers, our data will be connected across computers. Transparency is another thing that helps . This is just the logical separation between users and their actual in person devices. Finally we have efficiency. AWS helps us achieve optimal resource utilization.

* **Security**: <Security is a must-have for the client. Explain how to protect user information on and between various platforms. Consider the user protection and security capabilities of the recommended operating platform.>

AWS offers identity and access management. This allows for the secure access to your AWS resources. AWS has a web app firewall, which protects web applications from different exploits. We have encryption for data at rest and in transit. We have AWS CloudTrail, which will monitor and log account activity. Lastly we have Amazon Guard Duty, which gives us threat detection, that finds potential threats real time.