The Gaming Room

CS 230 Project Software Design Template

Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
3.0	04/21/24	Kevin Erdogan	This revision includes recommendations for The Gaming Rooms on various facets of security, storage and memory management.
2.0	04/07/24	Kevin Erdogan	This revision includes a more thorough write up and hardware and software recommendations for The Gaming Room's backend system.
1.0	03/23/24	Kevin Erdogan	This revision includes all the initial writes-up for the Software Design Documents, including all requirements and summaries below.

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 Gaming Room wants to develop a web-based game that servers multiple platforms based on their current game, Draw It or Lose It, which is currently available on the Android App Store. But they lack the technical knowledge to set up an environment and facilitate the development of the web-based, platform independent version of their Draw It or Lose It. A proposed solution would be using various separate systems and a web-app framework designed for platform responsiveness to achieve a platform independent experience on the web.

Technical Requirements:

- The application must be platform independent and be able to accessed on any device that supports modern Web frameworks
- The application must be under a Web Framework that prioritizes responsiveness, fast rendering, and efficient state change.
- The application should have a backend server to manage all various instances of Draw It or Lose It simultaneously.
- The application's backend server must make sure that there can only be one instance of a particular game in memory.
- The application's backend server must be able to handle multiple teams in one game and must be able to handle multiple players per team.
- The application must make sure that Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
- The application should render images from a large library of stock drawings
- The application should be able to render drawings at a steady rate (under the same frame rate)

Business Requirements:

- Enhanced User Experience.
    - The game application should provide an engaging and seamless experience for users as if they were playing it natively on their device. The application should allow them to easily create and join teams, as well as participate in gameplay without running into issues due to bugs or platform inconsistencies.
- Security
    - The game application, including the backend server, should provide security through the form of encrypted traffic using SSL, adding a rate limiter for an anti-DDos system, and ensuring type safety with various exception handling.
- Adheres to Game Rules
    - The game application should follow all of the rules of Draw it or Lose It, such as constraining users on a time limit and keeping count of the rounds and time. The application should handle game flow properly, picking the next player property and awarding teams with points and keeping general score.

Design Constraints

- **Always Online**
  - Due to the nature of the web-application, the user's device must be connected to the internet at all times, especially during gameplay. This constraint forces us to consider our user's internet connections, how stable and fast they are and adjusting our systems to accommodate packet loss or periodic loss of internet. We would have to design various methods into mitigating conflict because of packet loss and allowing the user's to "rejoin" their game if their Internet goes out.
- **Not All Devices Render The Web The Same**
  - Even though the web has became very standardized with every major browser basically supporting all modern features of CSS, HTML5, and ES6 Javascript, some browsers and platforms do not respond the same way. For instance, mobile devices lower the priority of Javascript's "useAnimationFrame" which could lead to significant problems for animations or any thing that renders based on the device's own current framerate. This constraint would need to be handled properly and code should accommodate various devices and platforms.
- **Responsiveness**
  - Due to the sheer domain of devices accessible to the web, it is mandatory that the entire web-application be responsive to all common viewport widths, ranging from wide desktop screens to small mobile devices. This constraint forces the application and it's UI design to accommodate all various viewport sizes while keeping all integral parts of the game intact.

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 class depicts various classes and relationships necessary for the Gaming Room web-applications.

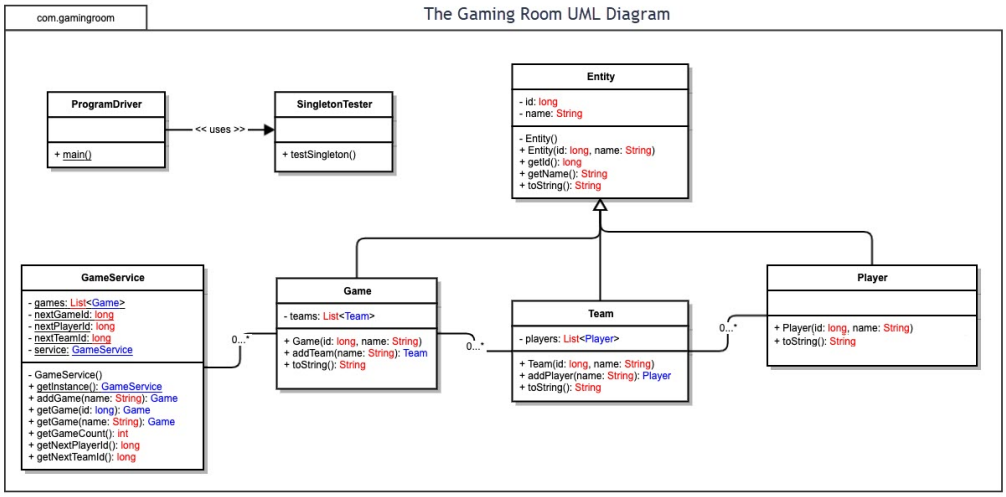
Classes:

- **ProgramDriver**
- **SingletonTester**
- **Entity**
- **Game**
- **Team**
- **Player**
- **GameService**

Relationships:

- The application starts with **ProgramDriver** which directly uses **SingletonTester** to test the application, these two classes and relationship serves to be a unit test for the software, making sure the fundamentals of the pattern and structure of the application passes a "test."
- **Entity** is the parent class of ***Game, Team, and Player***. These three classes inherit directly from entity, these three classes exist as Entities in memory; an **Entity** contains attributes that allow it to be uniquely identified. This implementation is very important for the software due to it's satisfaction of a technical requirement of all ***Games, Teams, and Players*** requiring it to be unique; while **Entity** does not ensure that each existence of itself will be unique, it does contain methods and variables that support Unique Identifiers to be implemented.
- **GameService** is a singleton class that houses all of the methods required to efficiently manage and distribute control throughout various games. **GameService** contains variables that houses all current games and forces Unique Identifiers to be assigned to ***Games, Teams, and Players***. On top of that, **GameService** directly uses the ***Game*** class, which it stores it inside of a ***List*** inherited Object; this storage in memory allows direct, fast access to any of the concurrent games running.
- **Game** is a child of the Entity class where it directly uses the ***Team*** class to store various teams within a game, which is directly apart of the business requirements of the game.
- **Team** is a child of the Entity class where it directly uses the ***Player*** class to store various players within a particular team. Which is also directly apart of the business requirements of the game.
- **Player** is a child of the Entity class, it has no real distinction from the ***Entity*** class, however **Player** exists to serve as a *type* in code to ensure type-safety and code legibility.

UML Class Diagram:



Evaluation

Development Requirements	Mac	Linux	Windows	Mobile Devices
Server Side	Mac OS would not be a viable platform for hosting a web-based software application, while it bases its underlying kernel as UNIX which is preferable for server hosting, it simply is not designed to host web-based software applications, it would be horribly inefficient and impractical to do so. The operating system is strictly made to integrate with the Apple Ecosystem and provide a very friendly user experience. There is no licensing costs to owning a Mac OS server but Apple's line of Server Machines has been discontinued for a long time.	Linux is a very server friendly operating system. It includes full control and customization of various modules and software for any specific needs. It is secure and safe for the operating system. Although, Linux is tricky to use due to its major focus on the command line, it also introduces an issue of "giving too much power to the user." However, it is important to note that Linux is the main operating system that many SaaS companies use, such as Azure or AWS. And on that note, Docker and Linux go hand in hand (but it is possible to have a Docker Image of Windows.) Other then RedHat, there really isn't any licensing costs to owning a server running on Linux.	Windows has been known as a server operating system for some time, since Windows NT was released, it gives an experience that many people are already familiar with, many normal Windows Applications still work, and things are easily configurable. However, Windows Server does cost money, while Linux typically does not (depending on the distribution.) Windows is also known to be slower and less secure than Linux and the amount of customization is limited. Windows has a steep cost to licensing for server use which ranges from $501 to $6,155 dollars!	Mobile Devices are cheap but not powerful, not only are they simple ARM chips, but they also typically cannot be customized to run unsigned code (such as the iPhone), let alone host a web server. Mobile Devices are too instable for a server context as well and would fare awfully in a server context. Most mobile devices don't have native support for Ethernet so much of the traffic would go through Wi-Fi. To my knowledge there isn't software that acts as web server like Apache or Nginx on mobile devices. If they were, they would be unoptimized and buggy. Mobile devices hold no license requirements nor costs due to the nature of mobile devices not being used for server computing.

Client Side	Really the only major thing to cater to Mac OS is adding safari support. Safari has been known to render things strangely compared to other browsers like Firefox or Google Chrome. It doesn't take much expertise in the Mac OS Eco-system to debug and develop web apps that works on Mac OS, however it is important to know that it does cost a decent amount due to the fact you can only run Mac OS on a Apple Computer.	There isn't much considerations for Linux, as it is free and requires not so much experience to setup and debug on. Linux uses the major browsers like every other platform and displays text similar to Mac OS; basically Linux works very well on the web and usually any bugs experienced are because of the user's configuration (for instance out-dated drivers)	Windows does cost money and has some additional steps to set it up for debugging mode, but many if not all the most popular browsers are supported on Windows and many frameworks are made with Windows in mind. It has a very low bar for experience.	Mobile Device considerations are some what decent. Browsers do render things differently depending on the device's operating systems (Android and iPhone), and it takes some expertise to properly debug web pages on those devices. Firefox has a feature where debugging on a phone can be synced with a computer, but iPhone seems to be the most challenging. Without having an iPhone, debugging an iPhone version of the web app would require a Mac Device to emulate iOS and have access to developer features. The cost would be unnecessary as the developer would need access to a Mac Device.
Development Tools	If the Server is chosen to be on Mac OS, some languages can be used: Swift, Objective-C, Java, Javascript, and C++. Mac OS has it's own XCode IDE to write native Swift and Objective C code but the options of Vim, Visual Studio Code, and IntelliJ are present	Linux is very versatile in-terms of what programming languages it can use: C, C++, Javascript, Java, Ruby, GoLang to name a few. The available editors are limited as many big name companies do not find the need to package their software for Linux, however	Windows has many options compared to Mac OS, many IDEs are mainly built for Windows such as Visual Studio, Visual Studio Code, Atom, IntelliJ, Sublime Text, Neovim, and others. Windows has the option to run the same languages as Linux but at a cost of	Only a handful of IDEs are available for mobile devices, to name the few is XCode and Android Studio Kit. These two are for iPhone and Android respectively. However, iPhones are very limited to Macs for development, while there is some workarounds such as React Native or Flutter which builds to both

		the common IntelliJ, Eclipse, Neovim, Emacs, and Visual Studio Code is completely available. Linux has the ability to easily virtualize with Docker and it may be a heavily viable option. I believe the only tool to deploy for Linux consistently and on a server context is Docker, Docker makes it really easy to bundle software and let it run anywhere, however bundling software for Linux is much more difficult as many different Distributions use different package managers.	performance and speed. Bundling software is very easy for Windows as many premade suites exist for the operating system to easily compress and bundle modules together to release a full solution. For a client-side deployment, it's very easy and optimal, but for a server context it becomes needlessly complicated.	Android and to iOS. Deployment to both platforms involve using its respective App Store which comes with various fees. For the Apple App Store, it costs a $100 dollars per year to upload Apps (which go through a verification process) and Google Play requires a one-time fee of $25 dollars.

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**: An appropriate operating platform that would allow the Gaming Room to expand Draw It or Lose it would be one of the many flavors of Linux, mainly anything Debian flavored. Linux would not only allow the costs of server computing to be low, but the dedicated resources Linux gives to programs allows for applications to run as optimally as they can. Many server applications are made with Linux in mind, so things pretty much ship out of the box. We can use various systems such as Docker to ship out more features for Draw or Lose It; we can configure Web servers and internal proxies to allow a seamless experience for all clients on all platforms.

2. **Operating Systems Architectures**: We have two choices when it comes to Linux architectures, we can use the good old reliable 80x86 architecture or we can use the more experimental ARM architectures; these two architectures are dependent on the path. In my personal experience, SaaS services not only use ARM but also make it hard for real expansion and growth without hefty costs, and there's many businesses and companies that are stuck on SaaS platforms like AWS. 80x86 architectures usually require an actual machine, and usually takes up more power and generally costs more to own and maintain. They are more powerful in their own respect and the majority of applications are already written for 80x86. My suggestion is to stick with 80x86 as that many applications already exist for that architecture, it has the power we need, and the cost won't cause much overhead.

3. **Storage Management**: I generally believe that storage is important in any application. Retaining user data and logs is very important for a proper operation of software and business. To achieve this we can use a relational database such as MySQL or Postgres and use an abstracted backend to communicate with the databases. For physical storage we should use a RAID 10 setup with Drives rated to be used on Servers, which should handle years of constant use.

4. **Memory Management**: Linux is very friendly with memory and shouldn't have any overhead with any of the applications. The problem with a Windows based server is that Windows typically eats memory up and leaves very little for the actual application to use, which can cause slow-down, undefined behavior, wasted power consumption, and even crashes. The software itself should be mindful of memory use because serving hundreds and thousands of clients requires many instances in memory; being smart in when to dispose of memory is crucial.

5. **Distributed Systems and Networks**: In my experience, every server operator should plan on using tunnels for their connections, a great platform to use is Cloudflare. Cloudflare also bundles into security as well. The backend should run under a SSL proxy that easily encrypts all the data between the open wire so that all user – server interactions are encrypted. A CDN (Content delivery network) can be used to deliver reliable assets and information as fast as possible. Various systems running in different areas of the world should also be used, however this venture can easily topple the cost into steep areas.

6. **Security**: Despite the recent “xz-utils” backdoor scare from the Linux community, Linux is notoriously one of the most secure operating systems out there. This is because it is open-source and everyone in the world can take a look at the source code and look for any bugs or anything malicious. To protect user information, we can use an encrypted drive and a tightly secure SQL server to make sure that things are kept safe. Many layers of authentication will be required and we can use “the principle of least privilege" to make sure that user accounts can’t be compromised and reek havoc in the system.