**Draw It or Lose It - Platform Evaluation and Software Design CS Module 5: 230 Project 2**



**Author:** Marissa Lanza  
**Date:** 2/4/2024

Project 2 Module 5

**Document Revision History:**

| **Version** | **Date** | **Author** | **Description of Changes** |
| --- | --- | --- | --- |
| 1.0 | 1/21/2024 | Marissa Lanza | Initial version |
| 1.1 | 2/4/2024 | Marissa Lanza | Added Evaluation section |

**Executive Summary:**

The Gaming Room is expanding its Android game, Draw It or Lose It, to a web-based platform that supports multiple operating systems. The project involves multiple teams and players, emphasizing the importance of unique game and team names. Challenges include managing a large library of stock drawings and ensuring a seamless gaming experience across platforms.

**Design Constraints:**

The project faces challenges with distinct software development kits for Android, iOS, and web platforms. The API must accommodate three platforms and support one or more teams. Key considerations include ensuring unique game and team names, providing provisions for team captains to resolve conflicts, and implementing unique IDs for games, teams, and players.

**System Architecture View:**

The system architecture comprises essential classes such as ProgramDriver, SingletonTester, Entity, GameService, Game, Team, and Player. Principles of inheritance and associations are demonstrated, with the Singleton pattern ensuring one instance of GameService. Clear relationships are established between classes, adhering to object-oriented programming principles.

**Domain Model:**

The domain model outlines relationships and attributes of key classes. ProgramDriver serves as the main driver class, SingletonTester tests the Singleton pattern, and Entity acts as a base class for Game, Team, and Player. Well-defined associations between classes facilitate a cohesive and organized representation of the domain.

**UML Diagram: Gaming Room:**

The UML diagram effectively illustrates the structure and relationships of key classes, emphasizing principles such as inheritance, association, encapsulation, and the Singleton pattern. It provides a clear visualization of the object-oriented design principles employed in the Draw It or Lose It system.

A computer screen shot of a computer

Description automatically generated

**Evaluation:**

**Server Side:**

**Mac:**

* *Hosting Capability:* Macs can be used as servers, but licensing is expensive. Hosting the website is feasible, but costs should be considered.
* *Licensing Costs:* Potential licensing costs for macOS server operating system.

**Linux:**

* *Hosting Capability:* Linux is well-equipped for web-based hosting and is cost-effective. Highly suitable for hosting the Draw It or Lose It web application.
* *Licensing Costs:* Typically no licensing costs for Linux server operating systems.

**Windows:**

* *Hosting Capability:* Windows servers are secure and easy to set up but come with licensing expenses. Appropriate for hosting, but cost considerations are essential.
* *Licensing Costs:* Potential licensing costs for Windows server operating system.

**Mobile Devices:**

* *Hosting Capability:* While mobile devices can be used as servers, they may lack the power for high-end operations. Suitable for development purposes rather than hosting a web-based application.

**Client Side:**

**Mac:**

* *Development Considerations:* Good SDKs for iOS development, but a MacBook is required. Development costs may increase, and expertise in Swift is necessary.

**Linux:**

* *Development Considerations:* Development time and expertise are crucial considerations. Using Python and IntelliJ’s Ultimate IDE can facilitate cross-platform compatibility.

**Windows:**

* *Development Considerations:* Expertise is a significant requirement. .NET framework is recommended for security. Costs associated with Windows development tools should be considered.

**Mobile Devices (iOS, Android):**

* *Development Considerations:* Native app development for iOS (Swift) and Android (Java/Kotlin). Alternatively, cross-platform development using frameworks like Unity (C++).
* *Cost, Time, and Expertise:* Higher development costs for platform-specific development. Longer development time due to separate codebases. Expertise in iOS and Android development required.

**Compatibility Across Platforms:**

* *Web Browsers:* Utilize responsive design techniques to ensure adaptability to different screen sizes. Regularly update and test the application on major browsers.
* *Mobile Devices:* Implement platform-specific design patterns for a native look and feel. Consider cross-platform frameworks for code reusability.

**Development Tools Evaluation:**

**Mac:**

* *Tools:* Coding on a MacBook using Swift and iCode. Licensing costs may be associated with Apple development tools.

**Linux:**

* *Tools:* Python comes pre-installed, and IntelliJ’s Ultimate IDE is recommended. Licensing costs for tools may vary.

**Windows:**

* *Tools:* Visual Studio Code is standard, supporting various languages like C++ or C#. Licensing costs may apply for Windows development tools.

**Mobile Devices:**

* *Tools:* Android Studio (Android), Xcode (iOS), or Unity for cross-platform. Costs depend on the specific tool and platform.

**Recommendations:**

1. **Operating Platform:** Recommend Linux Ubuntu Server on Kubernetes for efficient hosting.
2. **Operating Systems Architectures:** The Linux kernel ensures stability and security. Kubernetes facilitates easy system separation.
3. **Storage Management:** Prefer SSD storage for faster access. Use Kubernetes node for file storage and NoSQL node for game data.
4. **Memory Management:** Implement a watcher for system load to optimize memory usage and manage costs effectively.
5. **Distributed Systems and Networks:** Utilize Kubernetes for flexible and continuous game hosting in the cloud, providing accessibility to all operating systems.
6. **Security:** Implement a role-based security system for effective separation and access control.

This comprehensive evaluation addresses the unique requirements and challenges of each platform, ensuring a holistic approach to platform selection and system design for Draw It or Lose It. The recommendations provide a strategic roadmap for the successful implementation of the web-based gaming application.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac provides a stable hosting environment, and its user-friendly interface may appeal to certain developers. However, Mac's limited hardware compatibility and potential licensing costs might be considerations when scaling up to accommodate thousands of players. | Linux is well-suited for hosting web-based applications due to its robust server capabilities, stability, and cost-effectiveness. Its open-source nature allows for extensive customization and scalability, making it an excellent choice for handling thousands of players on a web-based platform. | Windows is a popular choice for web hosting, with a large user base and developer-friendly tools. Its scalability and performance can handle a significant player base, but potential licensing costs for Windows Server should be considered. | Mobile platforms may not directly host web-based applications, but they are crucial for supporting players on their respective devices. The application's responsive HTML interface should ensure compatibility across web browsers and mobile devices, providing a seamless user experience on Android and iOS. |
| **Client Side** | Cost would be low due to the HTML interface for the application; for Mac users, they would use their preferred browsers to access it easily. | Like the other computer OS’s, cost would be low; Linux users will use their preferred browsers to access the web-based application. | Desktop clients running Windows will use their respective web browsers to access the application easily. So, the cost to the client would be low. The existing Android application will continue to support players on Android devices through the native app; on the other hand, iOS does not directly support Android's native applications. To support Android users on a web-based interface, the responsive HTML design will allow players to access the application through their mobile web browsers. For iOS, an HTML interface is essential to enable iOS users to access the application. | Development considerations include programming languages such as C/C+ for Mac, C# for Windows, and Java for mobile devices. |
| **Development Tools** | A Mac Book that has iCode on it. All coding will be done using swift. | Python comes already installed on most Linux distributions. You could use IntelliJ’s Ultimate IDE to code for this. | Visual Studio Code is the standard and best way to code Windows applications. You could use relatively any language but C++ or c# is what most Windows programs are wrote in. | There are 3 options as far as mobile goes. Android’s you will need someone that specializes in Android Studio to develop the app. For iPhones you need someone that has a Mac book that can develop using swift in iCode. Or you can have someone develop the app using unity. Which is C++ and then can convert to either an android app or and iPhone app. But you will still need a mac to convert it to an iPhone app. |
| Development | -Programing | -Languages: C/C+ | -Languages: C/C#, | -Languages: Java, |