

<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 | 07/21/2024 | Raian Osman | Working on the blank template, filling in each section accordingly. |

**Executive Summary**

This document outlines the design requirements and architecture for The Gaming Room's web-based game "Draw It or Lose It". The application itself must meet the following requirements:

* Managing multiple teams (with unique names)
* Ensuring that each team has multiple players
* Maintaining one instance of the game in memory

**Requirements**

As we move forward, it's critical to understand The Gaming Room's technical and business requirements for the "Draw It or Lose It" web application. Technical requirements are the specific needs for the develoipment and functionality of the application. Business requirements regard the broader objectives of the project, such as user experience, game flow, and deployment. Please see the breakdown below:

**Technical requirements:**

* **Team management:**
* Application must support the creation of one or more teams.
* Allow users to check the availability of the game and team names in real-time.
* **Unique indentifiers:**
* Game, team, and player names must be unique.
* **Singelton pattern:**
* Ensure only one instance of the game can exist in memory at any given point.
* **Game Mechanics:**
* A game consists of four rounds, each lasting one minute.
* Drawings have to be rendered from a library of stock of drawings.
* Drawings take 30 seconds to render.
* A team must guess the puzzle within the one 1 minute time frame.
* If the current team cannot guess correctly the remaining teams get a 15 second window to offer one guess each.

**Business requirements:**

* **User Experience**: Providing an easy to use interface for end users.
* **Game flow**: Ensuring that the web app can handle image transitions, has a timer functionality to handle the rounds.
* **Deployment and maintenance**: Ensuring that the team has a deployment and testing environment. Making sure that the web app is scalable to handle an influx of traffic.

**Design Constraints**

Along with the business and technical requirements, it's also important to consider design constraints. With the game being a web app, the distributed environment will have the following design constraints:

* **Scalability**: The application needs to be capable of efficiently managing varying loads across different geographical locations. At any given point there will be multiple games (each having multiple users) and we must ensure that there will be no performance degradation.
* **Latency**: The web app will also need to be able to handle real-time interaction. This means that we will have to have CDNs and data caching mechanisms in place as well.
* **Security**: As each user will be creating an account, protecting their PII is crucial. The application must incorporate data encryption, secure authentication, and authorization protocols.

**System Architecture View**

The system architecture must include the following aspects:

System and Subsystem Architecture:

* **Tiers**:
* **Application Tier**: Host the core game logic, manage user sessions, handle API requests, interact with the database.
* **Presentation Tier:** This tier includes the user interface, which will be reached through the web browser via the internet.
* **Data Tier:** This will include the relational databases to store user information, game data, session states, and anything else related.
* **Components**:
* Web server
* Appliation server
* Databases
* Client interface

Logical Topology:

* **Communication Flow**: Communication started with the client and they send requests via the internet and it's routed through the web server to the appliation server.
* **Networking:** The network architecture will include load balancers to distribute incoming traffic across multiple application servers, ensuring scalability and high availability.

Physical Components:

* **Servers**: Cloud-based VMs from providers like AWS will host the web and application servers. These VMs will be selected based on the required CPU, memory, and storage specifications to handle expected user loads.
* **Networking Equipment:** Virtual networking components, including load balancers and firewalls, will be configured within the cloud environment to manage traffic and secure the application. Again, using AWS to provision
* **Storage Devices**: Cloud-based storage solutions will be used for both databases and static assets, ensuring scalability and reliability.

**Domain Model**

The diagram below is a visual of the web application. We have the 'Game', 'Team', and 'Player' classes which will inherit from the 'Entity' class, showing inheritence and that they share common attributes ('id' and 'name').

'Game' and 'Team' both have a one-to-many relationship. 'Game' with 'Team' and 'Team' with 'Player'. This shows that each game can have multiple teams and each team can have multiple players.

We also see that each class has private attibutes ('id', 'name', and 'teams'). This protects the data and ensures it can only be accessed through defined interfaces. The design also follows the singleton pattern, ensuring that one instance of the game exists throughout the web app.

Overall, the UML showcases that the design fullfills the software requirements with the unique identifiers 'Game', 'Team', and 'Player'. The singleton pattern enforces that only one instance of each game can exist in memory, and team and player management.



**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** | Mac computers, while not typically used as servers in production environments, offer a stable and secure operating system for hosting web-based applications during development. | Linux is a popular choice for hosting web-based applications due to its stability, security, and flexibility. It supports a wide range of server software, including Apache, Nginx, MySQL, and more. | Windows Server is a platform for hosting web-based applications, particularly those built with Microsoft technologies like .NET and SQL Server | Mobile devices are not typically used for hosting web-based applications but rather as clients accessing these applications |
| **Client Side** | Mac users typically expect high-quality, seamless experiences that integrate well with other Apple services. | Supporting Linux clients involves ensuring that applications run smoothly across various Linux distributions. The diversity of distributions (Ubuntu, Fedora, Debian, etc.) can introduce compatibility challenges. | Developing for Windows clients involves ensuring compatibility with a wide range of hardware and software configurations. Windows has a large user base, making it essential to optimize for performance and security. | Developing for mobile devices involves ensuring compatibility with various operating systems (iOS, Android). |
| **Development Tools** | Mac development primarily uses Xcode, a comprehensive and powerful IDE that supports Swift and Objective-C. | Linux offers tools, including GCC, Clang, and various IDEs like Eclipse and VS Code. The command-line interface and scripting capabilities (Bash, Python). | Windows development utilizes Visual Studio, which supports a range of languages including C#, C++, and .NET. Other tools like PowerShell, Docker for Windows, and various package managers (Chocolatey, NuGet). | Mobile development uses specialized IDEs like Xcode and Android Studio, along with languages such as Swift, Objective-C, Java, and Kotlin. |

**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 an operating platform we'd build the web application on Linux due to it's stability, security, and scalability. Linux supports a wide range of programing languages and development tools to facilitate the development of the game.
* Scalability and Elasticity:
* **Auto-scaling**: Implement auto-scaling groups on AWS to automatically adjust the number of running servers based on real-time traffic demands. This ensures cost-efficiency and high availability during peak usage periods.
* **Load balancers**: Again, leveraging cloud technologies like ELB in AWS will help deploy our linux servers on Amazon EC2 and distribute traffic across all of our deployed instances.
* **Operating Systems Architectures**: Linux operating system architectures are based on modular designs, which allow for efficient resource management. Key components include:
* Kernel
* Shell
* File System
* Package Management System
* **Storage Management**: For storage management we can leverage a cloud provider such as AWS. AWS offers S3 for the static images, which we can use a persistent storage for image generation. Or we can use EFS to attatch to our servers for lower latency.
* **Memory Management**: Linux can use serveral memory management techniques to ensure the efficient use of the RAM, memeory, and CPU. We can use a combination of physical RAM and swap space. We can also use cache management to ensure performance by keeping the data "close". Again, we can use a cloud provider like Amazon ElastiCache.
* **Distributed Systems and Networks**: Communication between different servers will require a microservices architecture, meaning we break down the game's application into smaller independent services that can communicate via the network using APIs. To make sure the application isn't overwhelmed we can use a load balance like ELB to distribute traffic over our servers. For real-time communication we'd leverage WebSockets.
* **Security**: There are 4 consideration we'd to take to secure our application.
* **Encryption:** Use SSL/TLS for data tranmission both in transit and rest.
* **Authentication and Authorization:** Implement OAuth and role-based access so that only authorized users can access the game.
* **Firewalls:** We can use firewalls to protect our web application using cloud based services.
* **Data Protection**: Implement policies for data privacy, consistency, and protection.
* **Identity access management**: Give developers and other individuals their required permissions nothing more and nothing less.
* **Critical Information for the Client:**
* **Server Deployment**: Linux is recommended for server hosting due to its scalability, security, and cost advantages. However, if there is a preference for another platform, additional licensing costs and resource considerations must be accounted for.
* **Cross-Platform Development**: By focusing on a web-based approach, the application can be deployed across multiple platforms with consistent user experiene. This method reduces the need for separate development efforts for each operating system.
* **Development Tools**: Utilizing a combination of free and open-source tools, along with specialized IDEs for specific platforms, will optimize the development process and control costs.