

<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 | 09/20/25 | Keegan Williams | Completed sections for project 1. |

**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**

Based off of the information provided to me, it seems as though the company, CTS, has a game design ready. However, they need help making the environment for a web-based version of their game application. They have also given a list of their software system requirements for the actual game development.

**Requirements**

**Design Constraints**

Given all the above information, the constraints include:  
1. A singleton pattern per game instance, meaning there can only be one instance of the game running at a time within the memory of whatever machine it is on

2. Uniqueness of names, each game name and team name must be unique to prevent duplication

3. Unique identifiers, every game, team, and player needs a unique ID for reference for the system operations.

4. Team structure, the game needs to allow for one or more teams in order to play.

5. Platform constraints, the game design must be made independent of any hardware specifications.

**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**

**In the diagram I can see that the program driver uses the singleton tester as a part of main(). The game service class is in correlation with the game class, which then trails into the team and player classes. All three of the game, team and player classes are specified as entities within the diagram. Each of the classes provided have their own private and public operators which will come into play within the actual coding of the game and what can be accessed/changed and what cannot.**



**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** | Characteristics  Unix-based (like Linux) with a strong graphical interface and high stability.  Excellent for web development, especially for developers who need compatibility with Linux servers and access to UNIX command-line tools.  Commonly used by developers due to native support for Docker, Node.js, Python, and other web technologies.  Advantages  Compatible with many cross-platform development tools (e.g., Visual Studio Code, IntelliJ IDEA, Xcode).  Can run virtualized Windows and Linux environments.  Very stable for running local web servers (Apache, Nginx, Node, etc.).  Good security and developer support.  Weaknesses  More expensive hardware (MacBooks, iMacs).  Limited enterprise deployment compared to Windows in some organizations.  Certain server-side optimizations or frameworks may require adaptation from Linux-based environments. | Characteristics  Open-source and the most common server OS for web hosting.  Highly customizable, stable, and lightweight.  Supports all major programming languages and frameworks.  Advantages  Ideal for hosting web-based software (LAMP/LEMP stack: Linux, Apache/Nginx, MySQL, PHP/Python).  Free and open-source—low operating cost.  Strong community and excellent security.  Command-line tools and automation make deployment easy (e.g., via SSH, CI/CD pipelines).  Weaknesses  Steeper learning curve for new developers (command-line heavy).  Limited support for commercial software (e.g., Microsoft Office, Adobe products).  GUI support varies across distributions. | Characteristics  Most widely used desktop OS, also capable of hosting web applications via IIS (Internet Information Services).  Integrates tightly with .NET ecosystem and Microsoft Azure.  Advantages  Excellent support for .NET, ASP.NET Core, and C# web apps.  Large corporate adoption—easier for enterprise integration (Active Directory, SQL Server, etc.).  Many commercial-grade tools (Visual Studio, MS SQL Server).  Weaknesses  Less efficient for running open-source server stacks (e.g., LAMP).  Licensing costs can be high.  Historically less stable than Linux for long-running servers (though improved greatly with Windows Server). | Characteristics  Access web apps via browsers (Safari, Chrome) or as Progressive Web Apps (PWAs) and native hybrid apps.  Requires mobile optimization (responsive design, touch interaction, performance constraints).  Advantages  Access to wide user base—mobile-first design ensures usability.  PWAs and cross-platform frameworks (like Flutter or React Native) can minimize extra development.  Can integrate with device hardware (camera, GPS, etc.) via APIs.  Weaknesses  Smaller screens, less processing power.  Different OS ecosystems (iOS vs Android) require testing and optimization.  Deployment to app stores requires extra steps if packaged as native apps. |
| **Client Side** | Cost: is pretty high due to Apple hardware being required. TIme: medium as setup is straightforward with web stacks.  Expertise: Moderate to advanced familiarity with UNIX. | Cost: Very low as the OS is free and open source as is same with tools.  Time: Can be very time effective for someone who is familiar with Linux but slower for a new developer on Linux.  Expertise: Requires solid knowledge of command line and server configuration | Cost: Relatively high cost due to licensing and proprietary tools needed.  Time: Moderate as the use of Visual Studio can help simplify deployment.  Expertise: Intermediate as the developer will need strong microsoft ecosystem knowledge. | Cost: Medium to high depending on iOS developer account and apple product costs.  Time: High as there will need to be tests and optimization done for both iOS and Android.  Expertise: Requires responsive design knowledge and knowledge of mobile frameworks. |
| **Development Tools** | Languages: JavaScript, Python, Java, PHP, Ruby. Tools: Xcode, VS Code, Sublime Text, Homebrew for package management. | Languages: Python, PHP, Ruby, Java, JavaScript, C++.  Tools: VS Code, Eclipse, PyCharm, Docker, GIT. | Languages: C#, .NET, JavaScript, TypeScript, Python, Java.  Tools: Visual Studio, VS Code, IIS, Microsoft SQL Server, PowerShell, Azuer DevOps. | Languages: JavaScript, TypeScript, Dart, Swift, Kotlin.  Tools: Android Studio, Xcode, VS Code, React Native CLI, Flutter SDK, Chrome DevTools. |

**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**: Kubernetes on Linux nodes (Google GKE, Azure) running Ubuntu LTS. This platform is very portable with the use of containers across clouds so you can expand to almost any computing environment. This platform suggested is also scalable and have rolling updates.
* **Operating Systems Architectures**: Node OS such as Ubuntu LTS have control planes that are manages by cloud providers that handles the API server and controller. With namespaces and RBAC there are separate namespaces for devs, stage, and production with role-based access control. It also includes microservices for each logical function such as: authorization, matchmaking, game server and API all packaged as Docker images and displayed by K8s deployment.
* **Storage Management**: Relational data such as the game state and user accounts can primarily be stored with PostgreSQL while being managed by Cloud SQL and Azure databases. For static assets like images, audio and saved drawings the server can use object storage. For backups and archival needs, regular database snapshots can be exported to a cold storage and replicated object-storage backups across the DR.

* **Memory Management**: The Linux kernel provides a virtual memory, swap and memory overcommit policies that use tuned settings for production. For container-level it can use cgroups to enforce CPU and memory limit/requests for each container and define conservative requests. It will also use resource quotas at namespace level to prevent noisy neighbors. For the runtime level, Node.js tune depending on memory speed will will use clustering for multiple core usage.
* **Distributed Systems and Networks**: Through client and API connection using HTTPS for request and response flows. WebSockets such as Socket.io can be used for realtime gameplay for many players per game. For network componenets an ingress controller plus cloub LB for external access can be used. As well as edge locations or multi-regional deployments if player base is very large.
* **Security**: To protect user info across all platforms, the system should use TLS/HTTPS for all client-server and service-to-service traffic. Encryption should also be used in the form of DB encryption as well as RDS/Azure managed encryption keys. The system should also use OAuth2/OpenID connect for user aunthentication. Again, the use of RBAC should be used and attribute-based checls for in-app privileges.