

<Name-of-Software-Application>

**CS 230 Project Software Design Template**

Version 1.0

Table of Contents

[CS 230 Project Software Design Template](#bookmark) 1

[Table of Contents](#bookmark1)**2**

[Document Revision History](#bookmark2)**2**

[Executive Summary](#bookmark3)**3**

[Requirements](#bookmark4)**3**

[Design Constraints](#bookmark5)**3**

[System Architecture View](#bookmark6)**3**

[Domain Model](#bookmark7)**3**

[Evaluation](#bookmark8)**4**

[Recommendations](#bookmark9)**5**

[Document Revision History](#bookmark10)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 06/01/24 | Shawn Plaisted | Initial Version |

**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](#bookmark11)

The Gaming Room has requested a web-based adaptation of their Android game, "Draw It or Lose It.” The new version aims to support multiple platforms while maintaining unique game and team names and ensuring only one game instance exists in memory.

Requirements

* Support for multiple teams and players.
* Unique game and team names to prevent duplication.
* Ensure only one instance of the game exists at any time.
* Unique identifiers for each game, team, and player instance.

[Design Constraints](#bookmark12)

* **Scalability**: The application must handle multiple simultaneous users and games without performance degradation.
* **Concurrency**: Only one instance of the game should exist at any time, necessitating efficient management of concurrent access to shared resources.
* **Unique Identifiers**: Ensuring that each game, team, and player has a unique identifier to prevent conflicts.
* **Cross-Platform Compatibility**: The game must run seamlessly across different operating systems and devices.

[Domain Model](#bookmark13)

The UML class diagram provided represents the core entities in the game application: Game, Team, and Player. Each class inherits from the Entity class, which holds common attributes such as id and name.

* Entity: A base class for all other entities, ensuring each has an id and name.
* Game: Inherits from Entity, representing a single game instance.
* Team: Inherits from Entity, representing a team within a game. Contains multiple players.
* Player: Inherits from Entity, representing a single player.

1. Inheritance: All classes inherit from the entity class, promoting code reusability and reducing redundancy.
2. **Encapsulation**: Each class manages its attributes and behaviors, providing methods for interacting with these attributes.
3. **Singleton Pattern**: Ensures only one instance of game service exists, preventing multiple game instances in memory.

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

[Evaluation](#bookmark14)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac servers are robust but expensive, ideal for development and initial hosting. | Cost-effective and highly customizable. | Good compatibility but may have higher licensing costs. | Not typically used for hosting but can be used for testing. |
| **Client Side** | Requires consideration of Mac-specific development tools and user experience. | Linux requires familiarity with multiple distributions and potential variations. | Windows development is user friendly with a wide range of tools. | Requires different support for different OS versions. |
| **Development Tools** | Xcode and Swift | GCC and Clang | Visual Studios and .Net | Android Studios and Xcode |

| **Platfrom** | **Characteristics** | **Advantages** | **Disadvantages** | **Development Tools** | **Licensing Cost** | **Experience Required** |
| --- | --- | --- | --- | --- | --- | --- |
| **Linux** | Open-source, highly customizable | Free, stable, secure, large community support | Requires expertise, compatibility | HTML5, CSS3, JS, Electric, PWA | None | High |
| **Mac** | Unix-based, integrated development tools | Good security, integrated development tools | High licensing costs, less flexible | HTML5, CSS3, JS, Xcode, Electron, PWA | High | Medium |
| **Windows** | Widely used, extensive application support | User-friendly, excellent .NET support, strong community | High licensing costs, more vulnerable, | HTML5, CSS3, JS, Visual Studio, Electron, PWA | High | Medium |
| **Mobile** | Diverse ecosystems, wide user base | Cross-platform frameworks reduce development time, large market | Device fragmentation, different development environments | React Native, Flutter | None | High |

**Server Side Evaluation**

**Linux:**

* **Characteristics**: Open-source, highly customizable.
* **Advantages**: Free to use, highly stable and secure, extensive community support, and supports a wide variety of server applications and programming languages.
* **Weaknesses**: Requires expertise in Linux administration and potential compatibility issues with certain proprietary software.

**Mac:**

* **Characteristics**: Unix-based system, integrated development environment.
* **Advantages**: Good security features, integrated development tools (Xcode), and a user-friendly interface.
* **Weaknesses**: High licensing costs, less flexibility for server deployments, and limited community support for server applications.

**Windows:**

* **Characteristics**: Widely used in enterprises, supports many server applications and programming languages.
* **Advantages**: User-friendly interface, excellent support for the .NET framework and Windows-specific applications, and strong community and professional support.
* **Weaknesses**: High licensing costs, more vulnerable to security threats, and higher resource usage compared to Linux.

**Mobile:**

* **Characteristics**: Limited server capabilities; not typically used as a server platform.
* **Advantages**: Secure environment, strong application sandboxing.
* **Weaknesses**: Limited control over server configurations, high development and maintenance costs, not optimized for server performance.

**Client Side Evaluation:**

Ensure the application is compatible with all major browsers like Chrome, Firefox, and Safari and compatible with mobile devices.

**Development Process**:

* **HTML5, CSS3, JavaScript (React, Angular, Vue.js)**:
  + Used for building responsive and interactive user interfaces.
  + **Cost**: Free and open-source.
  + **Time**: Requires front-end development expertise.
  + **Expertise**: Knowledge in modern web development frameworks.

**Mobile**

* **React Native, Flutter**:
  + Cross-platform mobile frameworks.
  + **Cost**: Free and open-source.
  + **Time**: Reduces development time by sharing code across platforms.
  + **Expertise**: Knowledge in mobile development and the specific framework used.

**Development Tools**

**Programming Languages and Tools**:

* **HTML5, CSS3, JavaScript**:
  + **Impact on Development Team**: Standard web development skills required.
  + **Licensing Costs**: None.
* **React Native, Flutter (for mobile)**:
  + **Impact on Development Team**: Requires mobile development expertise.
  + **Licensing Costs**: None.
* **Electron, PWA (for desktop)**:
  + **Impact on Development Team**: Requires knowledge in desktop application frameworks.
  + **Licensing Costs**: None.
* **IDEs and Tools**:
  + **Visual Studio Code, IntelliJ IDEA, Android Studio, Xcode**:
    - **Impact on Development Team**: Different tools for different platforms may require team members to learn multiple tools.
    - **Licensing Costs**: Most tools are free, but some may have paid versions.

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**: I recommend using Linux for the server-side due to its cost effectiveness and flexibility.
2. **Operating Systems Architectures**: Linux supports various architectures making it easy to use for different server configurations.
3. **Storage Management**: Use a relational database like MySQL or PostgreSQL to manage game data, ensuring data integrity and efficient querying.
4. **Memory Management**: Linyux has multiple memory management techniques like virtual memory with limits memory leaks ensuring a smoother run program.
5. **Distributed Systems and Networks**: Implement a microservices architecture to manage different game components independently.
6. **Security**: Implement HTTPS for secure communication, use OAuth for authentication, and encrypt sensitive data both in transit and at rest. Regular security audits and updates are essential to protect user data.

Advanced Recommendations (23 JUN 2024)

1. **Operating Platform:** I still recommend using Linux based served. This is because their servers offer robust performance, scalability and security features. This makes them suitable for handling the demands of a multi platform game. They are also very widely supported, open source and cost effective.

2. **Operating Systems Architecture**: The Linux kernel is a monolithic kernel this means it includes all the core functions of the operating system like process management, memory management, and device drivers in a single large block of code running in a single address space. This allows high performance since all services run in the kernel space and can interact directly. Linux uses a preemptive multitasking model to manage processes, ensuring that multiple tasks can run simultaneously without significant performance degradation.

3. **Storage Management:** Use of modern file systems like Ext4 or XFS combined with RAID (Redundant Array of Independent Disks). Both Ext4 and XF5 are advanced file systems that provide high performance, reliability, and scalability. They support large file sizes and volumes, fast recovery from crashes, and various journaling modes to protect data integrity. Implementing RAID ensures data redundancy and improves fault tolerance. RAID 1 mirrors data across multiple disks, while RAID 5 uses striping with parity, allowing for data recovery in case of a single disk failure.

4**. Memory Management:** Linux uses virtual memory to abstract the physical memory, allowing each process to have its own address space. This helps in isolating processes and preventing them from interfering with each other. When physical memory is exhausted, Linux can use disk space as an extension of RAM through paging. If the system runs out of RAM and swap space, it can start swapping entire processes out to disk to free up RAM.

5. **Distributed Systems and Networks:** Implement a client-server model where the game server handles game logic, user data, and synchronization, while clients (running on various platforms) handle the user interface and interactions. Also use standard protocols like TCP/IP for reliable communication and WebSockets for real-time interactions between the game server and clients. Implement mechanisms for handling network outages and latency, such as retry logic, caching, and eventual consistency to ensure a seamless user experience.

6. **Security:** Using a combination of methods with give the best results,use TLS (Transport Layer Security) to encrypt data transmitted between clients and servers. We will also have to make sure to implement robust authentication mechanisms to verify user identities and control access to resources. It is crucial to ensure users sensitive data is protected, we will use a strong encryption algorithm to ensure this. Also use firewalls to control incoming and outgoing network traffic and intrusion detection systems (IDS) to monitor and alert on suspicious activities.