

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 2.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/20/2025 | Taras Matsuk | Revised draft Updated Summary, Requirements, Constraints, and other pertinent information needed for this project |
| 1.1 | 08/03/2025 | Taras Matsuk | Updated recommendations and Development tools |
| 2.0 Final | 08/16/2025 | Taras Matsuk | Updated information touching on archive needs for OS specifics. Updated on the implementation of architectures based client needs. |

## [Executive Summary](#_sbfa50wo7nsh)

The client wants to take their existing Android-only game, “Draw It or Lose It,” and bring it to the web as a multi-platform application. The goal is to build off the current Android version while getting help with setting up the environment and making development more efficient. This web-based version should be developed using a cross-platform framework. Each game session and team name must be unique, and only one active instance of a game should run at a time. Multiple players will be assigned to each team, and games can involve either one or several teams.

## Requirements

The system needs to support one or more teams per game, with each team containing multiple players. To enforce a single active game instance, unique IDs will need to be generated for each game, team, and player. Users must also be able to check if a chosen team name is already in use to ensure all names are unique.

## [Design Constraints](#_2et92p0)

The biggest challenge here is deploying across multiple platforms. Since not all team members are familiar with cross-platform tools, this might mean splitting development across teams with specific environment experience. Another major constraint is keeping everything consistent and aligned with the client’s functional requirements, regardless of platform.

## [System Architecture View](#_ilbxbyevv6b6)

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](#_8h2ehzxfam4o)

The Entity Class is the parent class of Game, Team, and Player classes. This means that these classes are, Entity’s child classes, will inherit Entity’s attributes, while each being assigned attributes of their own, that are separate to the parent class. The Game Service class handles the core functionality needed to meet the client’s requirements. It ensures only one game instance exists at a time and that game names, team names, and player names each have unique IDs. The Program Driver holds the main method and interacts with the Singleton Tester class. The Game class maintains a list of teams, and each Team class holds a list of players. The Player class does not hold any lists; instead, it ensures each player has a unique ID that can be assigned to a team. Although players belong to teams and teams contain players, the Player class itself does not store any reference to a team or game.

**"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](#_2o15spng8stw)

Updated information for each recommendation based on further learning. Alternatively added the system architecture view to include the model. The majority of this game could be coded in C++, as it appears to be the most popular language for most game developers. I found some info on server options that are cost-effective, including renting servers or using a community-based peer-to-peer (P2P) network. Games like *Minecraft* and *RUST* already use these models, which makes them worth considering. That said, I’m not familiar enough with either approach to fully recommend them yet, especially since I’m unsure about the potential security risks involved.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | **Advantages:** macOS servers can run macOS, Windows, and Linux applications side-by-side. They are supported with extensive documentation and reliable technical support, and the operating system has remained consistent over time, making it user-friendly for experienced users.  **Disadvantages:** macOS systems are expensive compared to Windows, Android, or Linux, and hardware options are limited. In addition, updated software versions often require new purchases or even new devices. | Advantages: Linux offers a wide range of distributions and is open-source, with many free or affordable solutions available. Its flexibility and ease of customization make it highly suitable for servers and embedded systems. Linux is also recognized for having stronger security protocols than macOS or Windows.  Disadvantages: Linux has limited availability of pre-built machines, occasional file format compatibility issues, and fewer resources detailing potential security weaknesses compared to other platforms. | Advantages: Windows provides extensive customer and software support, a wide selection of compatible hardware, and frequent, streamlined updates due to its large user base. For corporate users, it includes built-in authentication through Active Directory integration at no additional cost.  Disadvantages: Windows has weaker security protocols, making it more vulnerable to malware, spyware, and ransomware. It also offers limited mobility development compared to other platforms. | Advantages: Mobile platforms can efficiently manage server-side calls, optimize database queries, and provide persistent data storage. They also allow users and clients to access applications from virtually anywhere, while preventing direct code manipulation.  Disadvantages: Mobile devices are generally less secure and more prone to corruption, offer less storage capacity than traditional servers, and are at risk of overheating, which can reduce performance. |
| **Client Side** | **Advantages**: macOS is straightforward to navigate once you have experience with it, and its consistency makes it reliable for developers familiar with the system.  **Disadvantages**: It is limited to Apple hardware, which can be restrictive for developers who don’t use Macs, reducing accessibility and flexibility. | **Advantages**: Linux is affordable, highly customizable, and gives developers full control over the environment. Being open source also makes it easier and quicker to maintain.  **Disadvantages**: With that flexibility comes responsibility, as developers are largely responsible for handling security and support, unlike macOS or Windows, which include built-in assistance. | **Advantages**: Windows is widely available with versions at different price points to fit project needs. It provides strong technical support and better built-in security compared to Linux.  **Disadvantages**: Effective use requires familiarity with the system, and costs can increase depending on which features or licenses are needed. | **Advantages**: Mobile platforms offer a wide variety of affordable tools and applications, making development accessible and convenient. They also allow for mobility in deployment and access.  **Disadvantages**: Mobile devices lack the full processing power of desktops, and the diversity of mobile operating systems—often tied to specific devices—can create compatibility challenges. |
| **Development Tools** | **Primary Language & Frameworks:** Swift, iOS SDK, macOS SDK **Tools:** Xcode (includes simulator, debugger, and build tools) **Cloud Services:** Xcode Cloud (automated builds, testing, and collaboration) **Impact on Teams:** Development is most efficient within the Apple ecosystem. Teams must use macOS hardware to run Xcode and deploy to iOS/macOS, which often requires dedicated Apple-based resources. **Licensing Costs:** Xcode is free, but publishing to the App Store requires an Apple Developer Program membership (~$99/year per developer). | **Languages & Frameworks:** Flask (Python), Node.js (JavaScript), WebSockets, MySQL **Tools:** Docker (ensures environment consistency and cross-platform deployment), Docker Hub (prebuilt images for faster setup) **Impact on Teams:** Provides flexibility for rapid prototyping and deployment. However, the wide variety of toolchains may require specialized expertise in backend development, databases, and containerization. **Licensing Costs:** Most Linux tools are open-source and free. Costs may arise from premium hosting, managed databases, or enterprise-level Docker Hub subscriptions. | **Languages & Frameworks:** C, C++, C#, Python, JavaScript, Firebase, AWS integration **Tools:** Visual Studio (IDE with debugging and Git integration), Git for Windows **Impact on Teams:** Broad language support enables a single team to handle diverse applications across desktop, cloud, and web. However, some projects may require developers skilled in both native Windows and cloud environments. **Licensing Costs:** Visual Studio Community Edition is free for individuals and small teams, while larger enterprises need Professional or Enterprise licenses. AWS and Firebase offer free tiers, with additional usage-based costs for production workloads. | **Languages & Frameworks:**  **Native:** Java (Android), Swift (iOS)  **Hybrid/Web:** HTML, CSS, React, JavaScript, MySQL **Tools:** VS Code, IntelliJ IDEA, Xamarin (cross-platform C# development) **Impact on Teams:** Native development typically requires separate iOS and Android teams due to platform-specific tools and deployment processes. Hybrid approaches allow one codebase across platforms but may require performance optimizations. **Licensing Costs:** VS Code is free; Xamarin is included with Visual Studio. IntelliJ IDEA has both a free Community Edition and a paid Ultimate Edition. |

## 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

## Recommend Windows Server on Microsoft Azure as the main server platform. Mainly integrating cleanly with the existing Android build of Draw It or Lose It. This platform supports a broad toolchain (Xamarin/.NET MAUI, React/Node.js, Java), and offers first class CI/CD, monitoring, and autoscaling. Using Azure which keeps the path open for iOS and web clients without any backend changes.

## Operating System Architectures

## Windows uses a hybrid kernel (Windows NT) with clear separation of user mode and kernel mode, supported by the Executive (process, memory, I/O, security subsystems) and the Hardware Abstraction Layer. This architecture combines performance with modularity getting the user the following:

## Robust process isolation and access

## Driver and I/O stack (good for high throughput networking)

## Stable ABI and strong back compatibility for smooth updates

## Storage Management

## Adopting layered storage strategy on Azure:

## Static media (game images/clues): Azure Blob Storage (Hot tier for active content; lifecycle rules to Cool/Archive for older assets). Add Azure CDN for low-latency global delivery and bandwidth offload.

## Relational data (players, teams, matches, leaderboards): Azure SQL Database or Azure Database for MySQL (built-in backups, PITR, geo-replication).

## Caching: Azure Cache for Redis to reduce DB load and speed session/state lookups.

## Configuration & secrets: Azure Key Vault with Managed Identities—no secrets in code or CI/CD.

## Backup & DR: Geo-redundant storage (GRS) and automated DB backups with tested restore runbooks.

## Memory Management

## Windows Server:

## OS level: Virtual memory with demand paging and working-set management ensures each process gets isolated address space; the kernel balances RAM vs. pagefile under load.

## App level: Use managed runtimes with modern GC (.NET or JVM) and set container memory/CPU limits to prevent noisy-neighbor issues. Employ object pooling for hot paths, stream assets (don’t load full images in memory), and cache carefully with size/TTL controls.

## Runtime tuning: Monitor with Azure Monitor + Application Insights (heap metrics, GC pauses, allocation rates) and right-size instances via autoscaling rules.

## Distributed Systems & Networks

## Designing backend as containerized microservices on Azure Kubernetes Service (AKS):

## Protocols: REST/gRPC over HTTPS for services; WebSockets or SignalR for real-time gameplay events and team updates.

## Integration: Azure API Management as the gateway (rate limiting, auth offload, versioning); Azure Service Bus for reliable messaging between services; Event Grid for pub/sub where appropriate.

## Resilience: Health probes, rolling updates, horizontal pod autoscaling, retries with exponential backoff, circuit breakers, and idempotent handlers.

## Clients: Android (existing), iOS (Swift), and Web (React) all communicate with the same API/Gateway; use offline caching on clients for brief outages and graceful reconnection logic.

## Networking: Azure Front Door or Application Gateway + WAF in front; Azure Load Balancer inside the cluster; private endpoints to data stores.

## Security

## Making security first class end to end:

## Identity & Auth: Azure AD B2C (player accounts) using OAuth 2.0 / OpenID Connect with MFA; Azure AD (tenant) for admin/staff. Role-based access control (RBAC) enforced at API and data layers.

## Transport & Data Protection: Enforce TLS 1.2+ everywhere; encrypt data at rest (Azure Storage & DB encryption), and rotate keys via Key Vault.

## App & API Security: Validate inputs, use parameterized queries/ORM, secure CORS, and implement least-privilege service principals.

## Perimeter & Monitoring: Web Application Firewall, DDoS Protection, Defender for Cloud, and continuous logging/alerting with Application Insights and Sentinel for SIEM.

## Compliance & Ops: Secretless deployments (Managed Identities), CI/CD signing and policy checks, regular pen tests, and security playbooks for incident response.