

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 5/24/2025 | Frank Santori | First iteration |
| 1.1 | 6/10/2025 | Frank Santori | Added Development Requirements and refined the document |
| 1.2 | 6/20/2025 | Frank Santori | Added the Recommendations section along with references section and cited sources |

## [Executive Summary](#_sbfa50wo7nsh)

Draw It or Lose It is a fast-paced, image-driven game application developed for multiple platforms. The primary challenge is delivering high-resolution images rapidly while maintaining performance and usability across desktop, mobile, and potentially console platforms. This design proposes an architecture built on REST APIs, cloud storage for large assets, and optimized memory handling for smooth gameplay. The solution balances performance, scalability, and security while considering the diverse technical requirements of various operating environments.

## Requirements

The client requires a scalable, multi-platform game that can deliver up to 200 high-definition images during gameplay. The application must support mobile and desktop clients, run efficiently across operating systems, and include features for user account management, content updates, and secure communications. It must also provide monetization features such as in-game purchases and maintain high security standards for user data.

## [Design Constraints](#_2et92p0)

Design constraints include limited memory and storage on mobile devices, varied OS security policies, and the need for real-time image rendering. Additionally, cross-platform compatibility requires careful use of common frameworks and tools that support multiple operating systems. Server-side processing must be optimized to handle multiple simultaneous users while maintaining low latency and data consistency

## [Domain Model](#_8h2ehzxfam4o)

The application is driven by a main class that initiates the creation of games, teams, and players using a GameService class, which follows the singleton pattern. This ensures only one instance of GameService exists at a time. Its constructor is private, and it can only be accessed through the getInstance() method, which creates the instance if it doesn’t already exist.

Games are created through the addGame() method, which uses the iterator pattern to prevent duplicate game names before adding the game to a list. Similarly, addTeam() and addPlayer() use the same pattern to avoid duplicate team or player names within their respective game or team.

The Game, Team, and Player classes inherit from a base class called Entity, which contains protected id and name attributes. The default constructor is protected to prevent null object creation, enforcing object integrity through overloaded constructors.

This design demonstrates object-oriented principles like inheritance and polymorphism through class hierarchy and constructor overloading. Encapsulation and abstraction are used by restricting direct access to constructors, requiring objects to be created through controlled service methods.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac offers a stable and secure development environment with excellent support for web technologies and game development frameworks like Unity. However, MacOS is not commonly used for hosting web-based applications due to higher hardware costs and licensing restrictions. It is better suited for client-side development and testing rather than as a primary server platform. | Linux is highly recommended for server-side deployment. It is open-source, cost-effective, and widely used for hosting scalable web applications. It offers robust security, frequent updates, and excellent support for distributed systems and containerization tools like Docker and Kubernetes. | Windows Server offers solid integration with .NET, IIS, and Microsoft tools, making it a viable option for enterprise-level applications. However, licensing costs are higher than Linux, and it is less flexible for open-source technology stacks. It’s useful when targeting a Windows-heavy user base. | Mobile platforms are not suited for hosting server-side applications due to hardware limitations and inconsistent connectivity. They should be treated strictly as clients, connecting to a centralized, cloud-based backend infrastructure. |
| **Client Side** | Supporting Mac clients requires macOS-specific design considerations, particularly for distribution via the Mac App Store. Development may require familiarity with Swift or cross-platform frameworks. Testing and deployment tools are reliable but sometimes limited by Apple’s ecosystem restrictions. | Linux clients are less common in the general consumer market, so support may be lower priority. However, Linux is valuable for development, QA testing, and as a base for backend operations. Tools and frameworks for cross-platform support make Linux compatibility feasible. | Windows remains a dominant desktop platform. Development tools are mature, with strong support for GUI frameworks and game engines. Compatibility with web technologies and a wide user base make it an essential target platform. | Developing for mobile requires knowledge of both Android (Java/Kotlin) and iOS (Swift). Cross-platform tools like Flutter or Unity can streamline development. Mobile development requires performance optimization, responsive design, and rigorous UI/UX testing. |
| **Development Tools** | Relevant tools include Xcode for native macOS and iOS apps, and Visual Studio Code for web or cross-platform projects. Programming languages include Swift, Objective-C, and JavaScript. | Linux supports languages like Python, C++, JavaScript, and Java. IDEs such as Eclipse, VS Code, and JetBrains IDEs are commonly used. It excels in backend and DevOps development. | Visual Studio is the leading IDE for Windows development. Languages include C#, C++, and JavaScript. Unity and Unreal Engine also support game development on Windows. | Development tools include Android Studio, Xcode, and cross-platform frameworks like Unity and Flutter. Programming languages vary depending on platform: Java/Kotlin for Android, Swift for iOS, Dart for Flutter. |

**Recommendations**

1. **Operating Platform**: For deploying Draw It or Lose It, Linux is the most suitable operating platform. It is widely used in professional server environments due to its reliability, flexibility, and low operational costs. Linux supports an extensive range of open-source tools and frameworks, making it ideal for web-based applications and RESTful services. Additionally, Linux systems are highly scalable, allowing the game to grow in user base without significant infrastructure changes (Red Hat, 2023). Compatibility with modern DevOps pipelines also makes Linux an excellent fit for continuous integration and deployment (CI/CD), reducing development time and improving software quality.

1. **Operating Systems Architectures**: The application should use a client-server architecture with a service-oriented design. The backend, running on a Linux-based server, will manage game logic, image retrieval, and user session control. Clients, including web browsers, mobile apps, and potentially console interfaces, will communicate with the server through a RESTful API using HTTPS. This decoupled architecture allows independent development and updates of client and server components (Microsoft Azure, 2023). It also improves fault isolation, as issues on the client side won’t directly impact the server’s core functionality. Horizontal scaling can be implemented through load balancers and container orchestration (e.g., Docker and Kubernetes) to manage increased traffic (Google Cloud, 2023).
2. **Storage Management**: To manage the large library of high-resolution images (approximately 1.6GB in total), cloud-based object storage like Amazon S3, Google Cloud Storage, or Azure Blob Storage is recommended. These platforms provide high availability, redundancy, and fast global access to assets (Amazon Web Services, 2023). Storing static assets separately from the application server reduces backend load and simplifies version control. Caching frequently used images locally also improves performance. Lifecycle management tools can automate cleanup and archiving of outdated assets.
3. **Memory Management**: The application should implement lazy loading and resolution scaling to reduce memory use, especially on resource-limited devices. Recent or frequently used images can be cached in memory to enhance performance, while unused ones are cleared to conserve resources. Profiling tools should be used to monitor and optimize memory use, and hardware acceleration should be leveraged where possible (Mozilla Developer Network, 2023).
4. **Distributed Systems and Networks**: To ensure smooth cross-platform interaction, the app will use a distributed system architecture with centralized APIs hosted on cloud infrastructure. A content delivery network (CDN) should be used to distribute static files and minimize latency across global regions (Cloudflare, 2023). Load balancers and failover strategies will ensure reliability during traffic spikes or server outages. Real-time monitoring tools will track dependencies between system components and alert on connection issues or degraded service (Datadog, 2023).
5. **Security**: Security is a top priority. All client-server communication must use HTTPS to protect data in transit. User authentication will be handled using OAuth 2.0, with role-based access control (RBAC) managing user permissions (IETF, 2012). Input validation and sanitization will guard against common vulnerabilities like SQL injection and XSS (OWASP, 2023). Sensitive user data will be encrypted both in transit and at rest. Regular vulnerability assessments, secure coding practices, and activity logging will further ensure data protection and application integrity.

References:

Amazon Web Services. (2023). Amazon S3 – Scalable Cloud Storage. <https://aws.amazon.com/s3/>

Cloudflare. (2023). What is a CDN?. <https://www.cloudflare.com/learning/cdn/what-is-a-cdn/>

Datadog. (2023). Cloud Monitoring as a Service. [Datadog Cloud Monitoring as a Service Software In Depth Review - The CTO Club](https://thectoclub.com/tools/datadog-review/)

Kubernetes. (2024). Kubernetes Overview. [Overview | Kubernetes](https://kubernetes.io/docs/concepts/overview/)

IETF. (2012). The OAuth 2.0 Authorization Framework (RFC 6749). <https://tools.ietf.org/html/rfc6749>

Microsoft Azure. (2023). Design a multi-tier architecture. [Multi-tier web application built for HA/DR - Azure Architecture Center | Microsoft Learn](https://learn.microsoft.com/en-us/azure/architecture/example-scenario/infrastructure/multi-tier-app-disaster-recovery)

Mozilla Developer Network. (2023). Using Hardware Acceleration. <https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API>

OWASP Foundation. (2023). OWASP Top Ten Web Application Security Risks. <https://owasp.org/www-project-top-ten/>

Red Hat. (2023). Why Choose Linux for Your Infrastructure?. <https://www.redhat.com/en/topics/linux>