

**DRAW IT OR LOSE IT**

# **CS 230 Project Software Design Template**

Version 1.0

## 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 | 07/20/25 | Amanda Dunphey | Completed all required sections of the software design document. |

**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](#_sbfa50wo7nsh)

The Gaming Room has requested a scalable and secure game management application to support their web-based game 'Draw It or Lose It'. The solution is a Java-based application utilizing object-oriented design principles, including inheritance, encapsulation, and design patterns such as Singleton and Iterator. This application ensures unique identification and naming of game entities while allowing expansion across platforms.

## Requirements

Business requirements include support for multi-user game sessions, unique naming for games, teams, and players, and scalability to mobile and desktop platforms. Technical requirements include using object-oriented Java, implementation of Singleton and Iterator patterns, and deployment readiness for a distributed web environment.

## [Design Constraints](#_2et92p0)

Design constraints include ensuring cross-platform compatibility, lightweight architecture for performance on limited-resource devices, and modular design to support future upgrades. Security must be built-in from the start, and communication between distributed components must handle outages gracefully while maintaining data integrity.

## [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 UML diagram models the relationships between the GameService, Game, Team, Player, and Entity classes. Entity is the superclass for Game, Team, and Player, promoting inheritance and code reuse. The Singleton pattern is implemented in GameService to ensure a single shared instance, while the Iterator pattern is used to avoid duplicates in lists. Encapsulation is enforced with private attributes and public accessors, and polymorphism is seen in overridden toString 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)

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** | <Evaluate Mac for its characteristics, advantages, and weaknesses for hosting a web-based software application.> | <Evaluate Linux for its characteristics, advantages, and weaknesses for hosting a web-based software application.> | <Evaluate Windows for its characteristics, advantages, and weaknesses for hosting a web-based software application.> | <Evaluate Mobile Devices for their characteristics, advantages, and weaknesses for hosting a web-based software application.> |
| **Client Side** | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Mac.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Linux.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Windows.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Mobile Devices.> |
| **Development Tools** | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Mac.> | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Linux.> | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Windows.> | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Mobile Devices.> |

## 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**: <Recommend an appropriate operating platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments.>
2. **Operating Systems Architectures**: <Describe the details of the chosen operating platform architectures.>
3. **Storage Management**: <Identify an appropriate storage management system to be used with the recommended operating platform.>
4. **Memory Management**: <Explain how the recommended operating platform uses memory management techniques for the Draw It or Lose It software.>
5. **Distributed Systems and Networks**: <Knowing that the client would like Draw It or Lose It to communicate between various platforms, explain how this may be accomplished with distributed software and the network that connects the devices. Consider the dependencies between the components within the distributed systems and networks (connectivity, outages, and so on).>
6. **Security**: <Security is a must-have for the client. Explain how to protect user information on and between various platforms. Consider the user protection and security capabilities of the recommended operating platform.>

**Server Side**

Java and the Spring Boot framework will be used to build the server part of Draw It or Lose It. This includes a built-in Tomcat web server, support for RESTful APIs, and easier dependency handling through Maven. Linux-based servers will be used for distribution because they are stable, safe, and cost-effective.

Some important server-side tasks are:

* Hosting the code for the game and running game sessions.
* A MySQL or PostgreSQL database is a safe place to store user accounts, points, and game data.
* Using role-based access control with Spring Security to do authentication and permission.
* Setting up WebSocket endpoints to get information on the game in real time.

All interactions between the client and server will be encrypted using HTTPS/TLS. The server will also follow OWASP secure coding guidelines to stop injection, cross-site scripting (XSS), and cross-site request forgery (CSRF) attacks. Firewall rules and fail2ban will be used to stop intrusions, and systemd services will be used for automatic restarts and uptime tracking.

**Client Side**

The client side will be made to work on computers running Windows, macOS, Linux, and mobile devices.

The app will be made in JavaFX for desktop systems so that the user interface stays the same and it works with JVM-based environments. For user management, matchmaking, and leaderboards, the client will connect to the server through secure RESTful endpoints. For live game updates, the client will connect through WebSockets.

A companion app will be made with Flutter for iOS and Android mobile platforms so that players can draw, guess, and join games from their phones. The REST and WebSocket APIs will be used by this mobile client, so all platforms will have the same features. Touch input and adaptable layouts are two mobile-specific improvements that will make the user experience smooth. Before data is sent to the server, all clients will check it, and authentication tokens will be kept safely (in Keychain for iOS, Keystore for Android, and encrypted local storage for desktop clients).

**Development Tools**

Development will use a combination of IDEs, version control, build tools, and testing frameworks:

* **IDE:** IntelliJ IDEA or Eclipse for server-side Java development; Android Studio for Flutter mobile client development.
* **Version Control:** Git with GitHub for repository hosting, issue tracking, and collaboration.
* **Build & Dependency Management:** Maven for the Java server; Gradle for the Flutter mobile client.
* **Database Management:** MySQL Workbench or pgAdmin for managing the database schema and queries.
* **Testing Tools:** JUnit 5 for unit and integration testing of server components; Mockito for mocking dependencies; Flutter’s widget testing for mobile UI validation.
* **Static Analysis:** SonarQube and SpotBugs for code quality and vulnerability detection.
* **Deployment Tools:** Docker for containerizing the server application; Jenkins for continuous integration and deployment.

This mix of tools and platforms makes sure that Draw It or Lose It is safe, easy to manage, and scalable, and that the experience is the same on all devices and operating systems that are supported.

**Recommendations**

**Operating Platform**

A Linux-based server system, like Ubuntu Server LTS, is the best way to run The Gaming Room's Draw It or Lose It expansion. Unlike proprietary systems, Linux is very stable, has good security, and is less expensive. Because it is open source, it can be changed in many ways, and the fact that it works with Java and Spring Boot makes release easy. It's easy for this tool to grow with more players, and it works well with both desktop and mobile clients.

**Operating Systems Architectures**

It has a monolithic kernel architecture, which means that key services like process management, file systems, and device drivers run in kernel space. This makes the system very efficient. The POSIX-compliant environment makes sure that Java applications can be used on different computers, and the ability to have multiple users supports safe role-based administration. The modular package system makes it easy to make changes without having to wait for the system to go down. Headless server setups also cut down on resource use for better performance.

**Storage Management**

The Linux Logical Volume Manager (LVM) will be used with a strong relational database management system like PostgreSQL or MySQL to control the server's files. This method gives you choices for flexible disk allocation, easy scalability, and data backup. Tools like rsync or pg\_dump will be used to set up regular automated backups, which will be stored in safe off-site locations or the cloud. Access Control Lists and file permissions will be strictly followed to make sure that only approved services can read or change private data.

**Memory Management**

Linux's virtual memory management system uses demand paging and swap room to make the best use of memory resources. The kernel changes how much memory is given to active processes on the fly, and the Java Virtual Machine frees up memory that isn't being used through trash collection. For Draw It or Lose It, the JVM will be fine-tuned with certain heap size settings and garbage collection algorithms (like G1GC) to keep latency low even when the game is being used a lot.

**Distributed Systems and Networks**

The system will have a client-server structure, and RESTful APIs and WebSocket links will make it possible to play games in real time and talk to people on other platforms. NGINX or HAProxy will be used for load balancing to send new requests to all server instances equally. This will improve uptime and scalability. All data transfers will be encrypted using HTTPS/TLS, and the app will have retry logic and failover processes to make sure that game state is not lost when the network goes down for a short time.

**Security**

At both the platform and application levels, security steps will be put in place. The built-in firewalls in Linux (ufw or iptables) will block ports that aren't needed, and intrusion prevention systems like Fail2Ban will protect against brute-force attacks. For application-level security, Spring Security will be used for authentication and authorization. Bcrypt will be used to hash passwords, and common OWASP Top 10 flaws like SQL injection, XSS, and CSRF will be protected against. When user data is being sent, it will be encrypted with TLS 1.3. When it is being stored in a database, it will be encrypted with AES-256. Also, API keys and access tokens will be kept safely in environment variables and changed often.