

# 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 | 05/24/25 | Colin Nagy | Created executive summary, requirements, design constraints, domain model, evaluation, and recommendations. |

**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 a successful game called Draw It or Lose It. Currently, this game is only available on Android. They want to take this application from mobile to the web, to play on any browser. The gameplay consists of multiple teams of players guessing a word, with hints rendering over the 30 second duration in the form of pictures. If one team fails to guess correctly, the other teams have a 15 second chance to guess correctly, one guess per team. Each game is four rounds, a total of one minute per round.

## Requirements

* Full gameplay features
* Web based
* Multi-browser functionality
* One or more teams per game
* Multiple players per team
* Unique game and team names
* Unique game instances, via unique identifiers for each game instance, team, or player

## [Design Constraints](#_2et92p0)

* Cross-platform
  + The game must work for any browser on any operating system.
  + REST API to communicate via HTTP
* User interface
  + This is an existing android application. Therefore, the user interface must emulate, or resemble, the original mobile interface.
* Multiple teams, multiple players
  + Each team has more than one player
  + Each game has one or more teams
* Unique IDs
  + Unique game IDs
    - Only one instance in memory at a time
  + Unique team name, check other names to see if it is in use

## [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)

Beginning with the GamesService. This is a line of none-to-many relationships. Game instances are stored in GameService, Team instances are stored in Game instances, and Player instances are stored in Team instances. Players need not be assigned to a team, teams need not be assigned to a game, and games need not be assigned to a game service. Game, Team, and Player each inherit from Entity, which provides a framework for names and IDs.

ProgramDriver contains the main function, which shows this will be the driving function to run the program. The ProgramDriver uses SingletonTester, which tests to ensure the code functions as expected.

**"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** | Unix-based, stable, secure. High levels of support and developer friendly. Expensive hardware. | Open-source, easily customizable. Many variations. Lots of software and many tools available. Smaller audience, but secure. GUI and hardware limitations. | Supports most hardware and software, strong support. Stable, yet less secure. Widely documented. Frequent updates necessitate regular restarts for security reasons. | Portable, nontraditional input methods. Gesture-based, small screen. Wide range of hardware, many limitations. |
| **Client Side** | Wide audience leads to wide familiarity. High development costs, time requirements, and expertise requirements. | Free for all use. Difficult UI creates a more specialized audience. Wide range of releases means more complex development. | Higher license cost, standardized OS means decreased time and expertise requirements. | Wider array of devices and operating systems increases complexity and time requirements, while OS come free. |
| **Development Tools** | Node.js, JavaScript, Objective-C, and Python are widespread. NetBeans, Eclipse, Oracle are useful IDEs. | Full suite of IDEs, notably Eclipse, NetBeans, and VSCode. Most development tools have a version of Java they will run on, leading to high adaptability. | Programming languages notably used are C, C#, C++, Python, and Java. Most tools will work natively on Windows for ease of use. | Swift, Objective-C, Java, JavaScript popular. Device emulators available. IDEs include Visual Studio, Android Studio, Xcode, and Eclipse. |

## 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**: I recommend using Linux for its cost, adaptability, and full suite of customizable options which fit any developmental needs. The main downside is the difficulty of learning the platform. This is due in part to the limited audience. Further, Linux uses a text-based user interface, as opposed to the graphical user interface of competitors.
* **Operating Systems Architectures**: Linux is managed by the Kernel, which manages CPU, memory, and all other low-level hardware interactions. The Shell is the primary tool for interacting with the system. While not a graphical user interface, this text user interface is more powerful and robust. The hardware is the physical elements which comprise the machine.
* **Storage Management**: I suggest a cloud-based storage solution. This will provide a scalable solution and offload maintenance requirements, while providing integrated security and reliability, as no local emergency can hamper operations. As Draw It or Lose It is gaining popularity, I recommend Amazon Web Services, as it is the most comprehensive cloud-based service available.
* **Memory Management**: The Linux operating system is totally customizable, down to the level of the kernel. Manual management is a possibility. However, as the proposed project uses Java, most memory management and garbage collection will be automated. Therefore, it is unnecessary to manage memory manually, reducing required expertise and development time.
* **Distributed Systems and Networks**: A client-server relationship offers the best solution for a multi-platform application. By handling Draw It or Lose It’s logic on one centralized server, the need for multi-platform support is greatly reduced. The client must only send TCP/IP requests to the main server. Therefore, we minimize the amount of work required to accommodate all platforms, as well as maintain a light file size for memory-restricted devices. In addition, Amazon Web Services maintain redundancies and network distribution services. This creates security that prevents both outages and connectivity issues.
* **Security**: Amazon Web Services have an integrated security suite, ensuring all security concerns server-side are automated. We need only focus on client-side security. By implementing a simple code-verification system, where the client sends a code to be verified by the server, we can ensure only untampered clients can send and receive information. By implementing TCP/IP protocols, we can ensure communications are secure. And by implementing back-end driven architecture, we can ensure the game’s integrity.