

DRAW IT or LOSE IT

# **CS 230 Project 3 Software Design Template**

Version 1.3

## 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 |
| --- | --- | --- | --- |
| 1.3 | 08/07/2023 | Nick Burnette | Recommend operating platform. |

## [Executive Summary](#_sbfa50wo7nsh)

Draw It or Lose It is a web-based game where players guess what is being drawn. A game lasts 4 rounds with a minute per round. Drawings are rendered from a library of stock images as clues. The team playing must guess within 30 seconds, or the opposing team gets 15 seconds to try. The app must accommodate multiple teams, each team with multiple players assigned. Game and team names must be unique, so name checks are a part of time creation. Only one instance of the game can exist in memory at any time, so each game instance will get a unique identifier.

## Requirements

The game is required to be cross-platform, though it is currently Android only. Entity will be established for each game, team, and player running from the GameService. The main program driver will check for single class instances.

## [Design Constraints](#_2et92p0)

Technical Constraints are to be determined. Performance constraints are related to swift response from the app, throughput, and memory usage. By using still images, the needs are likely not too significant. There are security constraints to consider to ensure other gamers cannot hack a play session or the user's phone. The design should be scalable to handle a growing user base, look at data volumes or multiple servers. In order to integrate to other devices, specific API's and data formats may be adhered to. The project will undergo a Time and Budget constraint. The legal team will review for any regulatory or legal requirements the design must comply with.

## [System Architecture View](#_ilbxbyevv6b6)

By being cross platform on mobile devices, we will look at the use of shared codebase to be reused across the different platforms. We will implement an abstraction layer for device-specific features and APIs. There will be packaging and deployment mechanisms for platform specific files when ready to distribute.

## [Domain Model](#_8h2ehzxfam4o)

In the UML the point of entry is the ProgramDriver Class, which uses the SingletonTester class to ensure unique instances of the game. The GameService class has association with Game class, a relationship of zero to many allowing it to manage multiple games. The Game class has a similar association with the Team class, allowing multiple teams. The Team class has an association with the Player class, with another relationship of zero to many, allowing a team to have multiple players. Finally the Game, Team, and Player classes connect to the Entity class to access its attributes and 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.**

## 

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | macOS is primarily used for Apple computers, and has strong security. It also lends itself nicely to moble iOS as they are related through Apple. It is user friendly, but requires expensive hardware and a controlled economic system through Apple Store. | Linux is particularly flexible as it is entirely open source. It is highly customizable, but has the smallest user base. | Windows is the most widely used OS for home PC's. It has a vast user base with a wealth of development tools. It is more susceptible to security issues than macOS or Linux. | Mobile devices offer the widest user reach, as nearly everyone carries a mobile device with them at all times. App stores make products easy to find and install, and you have a dedicated set of features available like cameras, speakers, GPS, sensors. |
| **Client Side** | Software development is more affordable than Linux by being easier to begin. Mac has numerous tools and a strong community. Software development has the additional benefit of transfering towards mobile iOS. | There is limited commercial support, and it has an initially steep learning curve. Otherwise there is a dedicated community of linux contributors available. Getting started can also be difficult and more costly as different distribution packages exist. | Windows would be the largest gaming market by itself, with the most commercial support and available tools for development. Additional resources would need to be invested in security, adding to the cost. | Development on iOS would be simplified when coming from macOS. On the other side, moving to Android would add difficulty. Devs would have to accomodate all the various ways the app could be used; tablets or phones with all the different screen sizes, vertical or horizontal or both. The App stores can be difficult for meeting requirements, and overall technological power is limited. |
| **Development Tools** | macOS and iOS for app development. Xcode is the official IDE by Apple and is free to download but requires an annual Apple Developer Program subscription. Visual Studio Code has macOS support and is free. Homebrew for package managing, Cocoapods for dependency management. | C/C++, Python, or Java all available for development with Linux. Visual Studio Code is available and free. Ubuntu for package management. | C#, C/C++, Python, or Java all available for app development. Visual Studio is the official IDE by Microsoft, it is free for small teams but may require a paid edition depending on tools needed. Visual Studio Code is available on Windows and is free. NuGet for package management. | iOS for Apple, Kotlin or Java for Android. Xcode on mac, Android Studio for Android which requires a one time registration fee on the Google Play Developer account. Gradle for Android automation tools, CocoaPods for iOS tools. |

## Recommendations

1. **Operating Platform**: The Gaming Room product *Draw It or Lose It* is the kind of game that will find its best audience on the mobile setting, where users can be in a room together playing a party style game on their own personal devices. The two options for mobile are Android and iOS. My recommendation will be Android for the following reasons: Android has a larger global market share at approximately 70.89% (in the most recent qaurter of 2023 according to bankmycell.com). The primary language for Android app developement is Java, which is already familiar to our company CTS. While iOS does favor clients with expendable income, it may have a more challenging development cycle as using Swift or Objective-C is not previously familiar to CTS. Java is also easily transferable to other platforms for future expansion.
2. **Operating Systems Architectures**: Android is built on the Linux Kernal, managing all available drivers during runtime. iOS is built in four layers that interface with each other via Core OS, Core Services, Media Layer, Cocoa Touch.
3. **Storage Management**: Android uses SQLite for storage management. It uses minimal resources and is self-contained on the device. It is SQL-based which many developers are familiar with, and is open source. Of note, iOS uses Core Data, an object-oriented framework provided by Apple. This approach makes it easy for developers to work with without needing low-level storage details. Core Data also streamlines version updates. Assuming the recommendation for Android is approved, the remaining considerations will focus solely on Android.
4. **Memory Management**: On SQLite, frequent opening and closing of a database can consume network and server resources. A singleton method would be employed and shared across the users in-game. The game would query and index the picture files as JPEGs for their smaller file size, and it would Data Cache images that are used frequently on the user's devices.
5. **Distributed Systems and Networks**: *Draw It or Lose It* would communicate to other platforms through RESTful APIs and web services. These architectural styles allow the app to communicate over HTTP methods. Our Android built game will send HTTP requests to said APIs, and the APIs will respond using JSON and XML formats, which are the common formats used across platforms. A network library like Retrofit will simplify HTTP request handling. Maven can manage the dependencies in our distributed system. Health checks would be built-in to help identify outages or connectivity issues.
6. **Security**: Users of *Draw It or Lose It* will have their usernames and passwords encrypted and sent through the APIs Android provides for security. Android Keystore system can store them as cryptographic keys instead of plain text files. An authorization check will occur for the user. I do not recommend added biometric security unless a payment method or sensitive information is stored, because the user experience can be negatively affected while simply attempting to play a game for fun. To prevent outside malicious attacks, CTS will ensure that secure coding practices have taken place. Input validation and input limits can prohibit buffer overflows and code injections. Because the game is purchased through the Android Market, payment security is handled by the platform. Updates and patches will be performed regularly to address issues reported to us and found through security testing.