

# **Draw It or Lose It: Design Document**

Version 3.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/14/2021 | Nathan Anglin | Original Document |
| 2.0 | 11/25/2021 | Nathan Anglin | Revision 1 |
| 3.0 | 12/12/2010 | Nathan Anglin | Revision 2 |

## [Executive Summary](#_sbfa50wo7nsh)

Our client, The Gaming Room, wants our company to develop a web-based game that serves multiple platforms based on their current game, Draw It or Lose It for Android devices. We plan to create a web-based program in Java, which can be used on multiple platforms through use of a a Java application on each platform. Java is well known for its ability to run on almost any operating system. Other programing languages can be used, but we recommend Java across the board for ease of development, and time and money needed to develop multiple apps with different languages.

## [Design Constraints](#_2et92p0)

* A game will have the ability to have one or more teams involved. [We will include the ability to safely create and access teams which are stored in an array]
* Each team will have multiple players assigned to it. [We will include the ability to safely create and access players which are stored in an array]
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name. [When creating a new game or team, we will check against these arrays to see if the name exists, if it does not, create the team, if it does exist, simply return that there is already an instance in use.]
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player. [All games, teams and players have a unique id number, which assures a new instance cannot be created if that game, team, or player name already exists.

## [Domain Model](#_8h2ehzxfam4o)

In this UML diagram, we show the layout of the server-based application to create, store and manage active games, teams, and players. We have created a Game service class, which uses the singleton method to make sure there is always only one game service available. This class then uses private methods to store ID identifiers for each Game, Team and Player, so there can never be a duplicate game, team, or player with the same name.

The Entity class develops basic functionality for the Game, Team and Player classes, which allows only private access for changing the Id and Name. This functionality is then inherited by the child classes Game, Team and Player.

Through this structure we can access creation of these objects without the worry of accidentally changing values we don’t want to. The Singleton class Game Service assures there is only one of each instance created, and validates this when creating new Games, Teams and Players, by checking the database for previously created objects with the same name.

**"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)

We have decided that a REST API is the most adequate solution going forward. The REST API style of communication between the client and server sides of a program ensure that every packet of information is self-contained. The server does not need to know the previous state to make a decision. This is to make the program RESTful, or to use certain sets of requirements to make the architecture safe and consistent. This type of architecture uses HTTP transfer of information in a way that is stable due to each packet being self-contained, that way each instruction can be read by each client application without the need to know previous states from the server.

On the client side, each program using the same server must follow the same rules and layout and use of input and output from the server. In the case of our project, each platform must be able to read and perform tasks based on a predetermined layout of the program. A good way to do this would be to have a well detailed and explained design document, that can be shared and updated between teams. We want to make sure each client for each platform is running within the same constraints.

To add more users to the database, the client-side program needs to send a request to the server. From this point, the server can check and verify a new user, and assign that user a username and password, as well as a player name. Delegating this only to be executed by the server makes sure there are no duplicate accounts, and that no more than one instance of the account is present.

Other features I would personally add to the game app would be a way to see how many other players are online, and the ability to add or remove them from your team. These basic features will make the social aspect of the game much better and attract new players and increase player engagement.

If The Gaming Room asked for the game to be also added on consoles, for instance the Xbox and PS4, we have a strong foundation in a RESTful implementation of the games core abilities. Most of the programming would be the same, just adding functionality for the individual systems and app stores. Most of the development work would need to be security on the new consoles, stability on the respective operating systems, and the requirements to be listed on that consoles store. Many platforms require a certain level of safety, compliance with programming standards, and functionality, such as button presses and shortcuts.

Finally, when moving the code to any new platform, quality assurance and rigorous testing is necessary to make sure each platform is stable and interacts with the HTTP packets sent from and to the server in the exact same way. The only changes in the programs based on the platform should be related to how to accomplish each goal of the program relative to that platforms programming language, layout, and interface.

The following shows notes for each platform, and what we can expect from the development of your app on the following platforms, and our recommendations for each.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | While a good program for end users, Mac OS, or iOS is not particularly good for web hosting, due to high overhead used for the GUI. | Linux’s main pro is it free cost as an OS. It also has a very lightweight instillation. The cons are a less friendly user interface for adjusting the hosted games’ setting. | Windows is a robust platform both for client and server side. Cons are its cost and overhead lost due to the GUI. | Mobile devices, while a good platform for users, is not a good choice for hosting the server code. This option is not recommended. |
| **Client Side** | Java app development will run well with this platform, and can be shared among iOS, Windows, and Linux. Further tested will need to be done within Mac OS to check for any QA or compatibility concerns. Java does a good job at running on many OS’s. Development requirements for programming is a strong understanding of Java, and a moderate understanding of the Mac OS. | Java app development will run well with this platform, and can be shared among iOS, Windows, and Linux. Further tested will need to be done within Linux to check for any QA or compatibility concerns. Java does a good job at running on many OS’s. Development requirements for programming is a strong understanding of Java, and a moderate understanding of the Linux OS. | Java app development will run well with this platform, and can be shared among iOS, Windows, and Linux. Further tested will need to be done within Windows to check for any QA or compatibility concerns. Java does a good job at running on many OS’s. Development requirements for programming is a strong understanding of Java, and a moderate understanding of the Windows OS. | Java can be used for Android. Java can be used for uniformity of program code with iOS, but with an increased cost because of difficulty, but will save time . Swift or Objective-C programming language used for iPhone App, but in this case, writing with Java will save both time and money. Advanced knowledge of the applications development for iOS and Android are required for these programmers. |
| **Development Tools** | Main program will be web-based, so the Eclipse IDE will be used to develop code. This code will then be tested with Apple’s eAWT and eIO APIs for Java on Mac OS X | The Eclipse IDE will be used to develop code. This code will then be tested with Java SE Development Kit (JDK) and Java SE Runtime Environment (JRE) | The Eclipse IDE will be used to develop code. This code will then be tested with the Linux Java virtual machine. | We will rebuild the android app to create uniformity and create an iOS app for iPhone. Java will be used for both platforms using the Eclipse IDE, which will be developed to be in line with App Store requirements for both systems. |

## 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**: We recommend a “Serverless” server platform to run your game. Serverless is a cloud platform where you can buy space on a server system, paying by execution of code, rather than buying a complete server. This way the game can grow organically, and when more computing power is needed, it can be easily added without changing hardware and buying new servers.
2. **Operating Systems Architectures**: For an operating system, we recommend Linux to run the program server-side. While Linus is a cheaper option for an operating system, in that it is open source and registration fees are not needed, there are other reasons why it is a good choice. Linux is known for being very secure, stable, and having great flexibility to run only what we need to run, thereby reducing execution costs in a cloud computing, or serverless environment.
3. **Storage Management**: Storage management for the initial game code is recommended to be run server-side, as the high-quality images would be too large to store locally. User data, team data and other needed databases also should be stored server-side. Not only will this make the game a smaller install for end users, but it will save on executions on the cloud server for downloading larger game files. In this way, data for the game can be stored on the server for easier maintainability and will overall keep the games operation cheaper and more stable.
4. **Memory Management**: Memory management should consider max number of games, teams and players that can be created and used at the same time. With a max level of creations, we can predict behavior of the server with high workloads and keep servers up with less players if there is an influx of creation on games, teams and players, or a large number of games happening at the same time.
5. **Distributed Systems and Networks**: We recommend a web-based server to be accessed by individual apps on each OS. The master code will run remotely on a server, which is accessed over the internet directly by the applications. We have decided that a REST API is the most adequate solution going forward. The REST API style of communication between the client and server sides of a program ensure that every packet of information is self-contained. The server does not need to know the previous state to make a decision. This is to make the program RESTful, or to use certain sets of requirements to make the architecture safe and consistent. This type of architecture uses HTTP transfer of information in a way that is stable due to each packet being self-contained, that way each instruction can be read by each client application without the need to know previous states from the server.
6. On the client side, each program using the same server must follow the same rules and layout and use of input and output from the server. In the case of our project, each platform must be able to read and perform tasks based on a predetermined layout of the program. A good way to do this would be to have a well detailed and explained design document, that can be shared and updated between teams. We want to make sure each client for each platform is running within the same constraints.
7. **Security**: Back-up servers and data backups are necessary to save data and functionality during an outage. A system to slow or limit number of games played at once will be required to keep the web service stable. User validation, and bot checks, (such as Captcha image verification) will need to be used to stop any malicious creation of assets or games, helping prevent outages and attacks. Finally, each OS will need to be thoroughly tested for security bugs with the Java platform. All user information, such as names and personal information will need to be transmitted with encryption, and no user passwords or data can be stored in plain text format. Use of the Linux Operating System server-side will help us use the most secure tools to keep user data safe. With a RESTful API, we can limit traffic, and keep quarantined data that should only be given to those devices or users that absolutely need it.