

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.3

## 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 | 05/18/2022 | Ryan Niebla | Initial treatment through Domain Model |
| 1.1 | --- | Ryan Niebla | Updated based on initial feedback |
| 1.2 | 06/02/2022 | Ryan Niebla | Completing development requirements sections |
| 1.3 | 06/16/2022 | Ryan Niebla | Completing recommendation section |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room wants to develop a web-based version of its game “Draw It or Lose It” which is currently available as an android application. The game has multiple teams competing in guessing a slowly rendered drawing over four rounds. The drawings are to be rendered over a 30 second time limit and if the playing team does not guess correctly in that time period, the other teams have a 15 second period to guess.

* “Draw It or Lose It” must run on multiple platforms
* Must support multiple teams per game with multiple players per team
* Games and teams name must be unique
* Game Service must be singleton

“Draw It or Lose It” must run on multiple platforms. As defined in the UML diagram we must maintain a single ***GameService*** that can hold multiple***Game*** objects. The ***Game*** object should be able to hold multiple ***Team*** objects and the ***Team*** object must be able to hold multiple ***Player*** objects.

## [Design Constraints](#_2et92p0)

The project should be developed with the goal in mind of staying within the budget. We need to research and enable a way to render the image withing the time frame and need to find a webserver to host the application.

## [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)

***Entity*** is the parent object for ***Game***, ***Team*** and ***Player***. This means that all ***Game***, ***Team*** and ***Player*** inherit variables and methods from the ***Entity*** object. Inheriting the same variables across multiple child classes is a type of polymorphism as each variable shares the same name and type but has a different value based on the class which inherits it. This setup also shows abstraction and encapsulation as the original variables and method are abstracted away from even the implementors of the subclasses, as well as encapsulated by the use of setters and getters protecting the data. ***Player*** and ***Team*** have and aggregation relationship wherein each team has a player. The same relationship exists between ***Team*** and ***Game*** (each game has a team), and ***Game*** and ***GameService*** (the singleton GameService has a Game). The main method is housed in the ***ProgramDriver*** class which uses the singleton, ***SingletonTester*** for testing purposes.

**"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** | Using Mac for the server gives you the most secure option but it is also the most expensive and least readily available. Easily scalable. Another concern with Mac is compatibility. | Linux servers are the most readily available and the cheapest to set up with free option as well as EC2 instances through Amazon. Easily scalable. The biggest drawback is they are less secure than Mac. | Windows servers are also readily available and an EC2 can run windows for a higher rate. Easily scalable. They are more expensive than Linux and less than Mac. Least secure. | Mobile Devices are able to be used as servers through the use of third-party apps (Tinyweb server). There are no benefits to using a mobile device as a server and a very big drawback of loss of said device. Not recommended. |
| **Client Side** | Highest cost client as there are stringent requirements to run on Mac. Moderate amount of time and understanding due to those requirements and familiarizing with the OS | Lowest cost client because Linux is open source. Moderate to high level of time as understanding of Linux OS system is necessary. | Higher cost than Linux but lower than Mac. Lowest time and expertise with little time devoted to learning OS. | Mobile devices vary wildly. High time and high level of understanding as there are a multitude of versions of any of the mobile OS’s that have different specifications. |
| **Development Tools** | Supports multiple languages including but not limited to Java, JavaScript, C++, C#, HTML, Python, Ruby, Swift. IDEs include but are not limited to Microsoft Visual Studio, IntelliJ, XCode, Eclipse, Atom, PyCharm. Jenkins support for CI/CD pipeline. | Supports multiple languages including but not limited to Java, Google Go, C# Groovy, C++, Typescript, HTML. IDEs include but are not limited to Microsoft Visual Studio Code, IntelliJ, Sublime Text, Brackets, Atom, Eclipse. Jenkins support for CI/CD pipeline. | Supports multiple languages including but not limited to Java, Ruby, Python, C# Groovy, C++, JavaScript HTML. IDEs include but are not limited to Microsoft Visual Studio Code, IntelliJ, PyCharm Eclipse, Brackets, Atom, XCode. Jenkins support for CI/CD pipeline. | Supports multiple languages including but not limited to C#, C++, Java, JavaScript, PHP, Python, HTML. IDE;s include Andriod studio, Visualk Studio, Eclipse, IntelliJ, AppCode ect. Jenkins support on Andriod for CI/CD. Despite having access to these tools Development on a mobile device is not recommended. |

## 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**: After weighing the pros and cons of each I have decided on recommending a Linux machine for The Gaming Room. Mobile was never a choice in this and Mac is not a viable option so we are really left between Linux and Windows. I opted on Linux as it is open source and able to run whatever browser we need to test on easily. Linux is also the least expensive option and it with a multitude of readily available option it should be quick to set up.
2. **Operating Systems Architectures**: Linux is open source and as such, is highly variable. Systems include a Kernel, which handles the majority of the tasks of the OS. System libraries which access Kernel functions as needed. System Utilities which do specialize tasks and the Shell which is the portion which the user interfaces with the kernel. With Linux being open source, the shell can be anything from a robust UI to a command line interface. For the appropriate architectural pattern, I would recommend the client/server Pattern. Essentially our Linux machine with be the server which will take in requests from our clients and provides appropriate responses.
3. **Storage Management**: For storage I would recommend implementation of MongoDB. Mongo DB is a NoSQL database and can be leveraged to hold all our customers PPI (personal private information) suck as login credentials, usernames ect. We can also create a custom schema within mongo to hold our image files. For implementation I would recommend implementing Spring Boot on our java project wich has built in features to make connect to a data base easier.
4. **Memory Management**: The memory for the app will be largely focused on the retrieval and loading of images from MongoDB. Once an image is retried it will load in memory When the match is over there will be a server-side check to see if the image is being actively used on another match, if it is the image will remain in memory, if not it will be released. Linux itself, being highly customizable, has multiple options to configure its memory settings which can be accessed through /proc filesystem. The basics of the Linux memory management system are Linux uses Virtual memory and physical memory. The physical memory is the systems memory while the virtual memory is a non-existent amount that the Kernel refers to. When an application loads in Linux the virtual memory is used to essential point at a memory location that will be used by the application. Once this virtual memory is allocated it cannot be allocated anywhere else untill released. Because the virtual memory is a non-existent amount that point to real memory, you do run the risk of over allocation which is something we need to consider in planning the memory management for The Gaming Room.
5. **Distributed Systems and Networks**: Each of our clients will be connecting to our server for this web-based application using a web browser of some sort. In order to communicate back and for the with client I recommend using REST api. Representations State Transfer (REST) is an architectural style that uses HTTP to get and receive requests. Our clients will use their web browsers to send HTTP requests to the server which have REST methods such as POST, PUT, GET, DELETE ect attacked to them. Once the server received the request the JSON that is attached can be translated and acted on. Then the server would send an appropriate response in terms of HTTP codes 404, 400, 200 ect to inform the client of the status of their request.
6. **Security**: Security is of paramount concern so we will tackle this with a multifaceted approach. First, we will make use of JSON Web Tokens for authentication of our users. We will implement a route guard for redundant security and to ward off someone from manually entering a url and accessing a restricted page. We will run scans on all OSS (open-source software) that we are leveraging in our application and will remediate any issues that are present. I would recommend implementation of Blackduck to run scans before pushing code to production environments. I would also recommend implementing a linter on all projects to identify code smells and would recommend running periodic Fortify scans (static applications security testing) to identify additional issues.