

Draw It or Lose It

# **CS 230 Project Software Design Template**

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 | 01/25/2022 | Andrei Kourouchin | Create initial software design template with information as to initial build. |
| 2.0 | 2/05/2022 | Andrei Kourouchin | Added information to evaluation as well as evaluations. |
| 3.0 | 2/20/2022 | Andrei Kourouchin | Added additional information regarding recommendations as well as potential concerns with security |

## [Executive Summary](#_sbfa50wo7nsh)

The application Draw It or Lose it has been presented to the development team with a specific goal in mind, setting up the environment for web-based gaming. This application will run in a web-based application. We will outline the design constraints and other recommendations to be made regarding operating platforms, memory and storage considerations and security.

## [Design Constraints](#_2et92p0)

As this is a web-based environment, we need to ensure that the software we are creating is light enough not to cause issues on a vast array of different devices, as hardware specifications will be significantly different across the board. We should also ensure that the language that we use will be able to be used by as wide an array of devices as possible, depending on the availability, this can reduce the number of users. Creating our data structures and deciding whether we’re going to store player, team and game information locally on the user end, or if we will store it in some sort of ADT on the back-end and provide information as needed to the user.

## [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)

There are a few OOP principles at play here, specifically inheritance which can be seen by the Entity superclass that is extended by the Game, Team and Player classes. We also demonstrate encapsulation by using public methods that affect private objects to prevent other classes from altering them. We also use abstraction here in order to show only the relevant information to the current operation. This can be seen by the addPlayer function which calls a getNextPlayer() function that provides a new Id. We do not do extra operations in each function, and they’re broken up logically so that if needed, we can change how one function operates without breaking the entire program.

There are a few classes here, but the main one is the Entity class which is responsible for creating the basis for our Game, Team and Player classes. The subclasses inherit from the Entity class, to prevent us from having to recreate several methods and variables again. The Three subclasses are linked between each other, as well as with another GameService class through a association where the relationship is optional, as well as can be associated with an unlimited number of instances of the other class. We also have a directed association between the ProgramDriver and SingletonTester where the ProgramDriver uses the testSingleton() function.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac has the capability of hosting a web-service game application, but with its relatively limited support, this is an unpopular choice. | Linux being the most flexible and having package managers will make development, rebuilds and upgrades relatively easy compared to their counterparts. Bash scripting also increases ease of use. Scalability and security are best in class. | Windows servers having some of the best support, as well as group policy that is incredibly detailed and relatively easy to use. Scalability is very good but is significantly more expensive than Linux servers. | Mobile devices should not be used for hosting a web server and have very limited capabilities due to their hardware. This can be done, however is not recommended. |
| **Client Side** | Java is fully supported by Macs, however they do mostly have relatively low limitations with hardware specifications, we should account for that when developing the client side application. | As Linux can be installed on all sorts of hardware and run varying types of software, we should ensure it works, however there are no serious considerations for a web-based program. | Windows being the most popular operating system should be the main one that we are developing for, as the most users will be using this one. Ensuring that everything works here is the most important. Java is fully supported. | Whether or not we use an application, or choose to use a web app, we will have to make certain considerations. Including hardware limitations, as mobile phones are typically underpowered compared to any other type of device. |
| **Development Tools** | Java and Eclipse are both fully supported by Mac, as well as what is likely to be a REST api that handles the web application communication. SQLPro studio can be used to manage databases like MySQL or Microsoft SQL server. | C++ and C IDEs like visual studio are typically not supported by Linux. As well as all other major windows-only applications. Developing for windows on a Linux machine means you need additional hardware for testing. However, Eclipse and Netbeans can be used on Linux for development. | We can use Visual Studio, Eclipse, NetBeans and a vast array of other IDEs on windows. When developing for windows, this operating system is preferable as testing can be done locally or in virtual machines installed on the same machine. | Android studio and Xcode are very common IDEs for android and MacOS development respectively. They both allow development in Java, c, C++. However Xcode also allows development in Ruby and Python. |

## Recommendations

1. **Operating Platform**: I recommend using a Linux server for the back end operating platform due to their high standards of security as well as low cost to set-up and maintain. The Linux operating system will also allow us to scale relatively easily compared to other types of operating systems which will help reduce load once there are many users on the platform.
2. **Operating Systems Architectures**: While this list is not in order, we should prioritize development based by market share, and open availability of the web app up as development continues. The browsers that we will be developing are as follows:

Chrome

Firefox

Edge

Safari

Safari (Mobile)

Chrome/Firefox (Mobile)

1. **Storage Management**: An instance of PostgreSQL is free and can be installed on a relatively low-cost Amazon AWS instance which can be scaled as demand and usage grows. This database has a high level of support and has been around for many years. Security features are available, however there should be no chance of a user having direct access to the database. There are not too many considerations regarding the actual size of storage for most operating platforms, as total size should only amount to ~2gb. We can use caching and preloading images on mobile to maintain a much smaller app size. Devices like laptops and Desktops have more than enough storage to handle this data size.

**Memory Management**: We can use several methods to optimize the way that the Linux operating platform uses memory. One of which is to cache information on the client-side when we can reuse the information later. Linux’s efficient data caching properties ensure that we do not have unnecessary processes running in the background. We can also explore image formatting to find the most efficient type of image, that can also be compressed for high transfer speeds.

1. **Distributed Systems and Networks**: As the web server and database are both hosted on AWS servers, we can ensure a relatively high uptime rate, as well as set up servers across different locations to maintain a level of redundancy that will prevent us from having many outages. Hardware limitations can also be mostly overcome by the fact we can spin up more servers and add to our infrastructure as needed. Secure connectivity for management can be ensured by SSH and VPN connections, as well as AWS software.
2. **Security**: Due to us choosing a Linux server as a backend, we have much higher security capabilities than running a windows server. We should also ensure that the application running on the client side never has direct access to the PostgreSQL database and is communicating through sanitized and authenticated input through the application running on the server. We will also implement role-based security to ensure that users have the lowest needed permission level to enjoy their game. Ensuring that the API has the least amount of communication available is also an important item to check as that opens the surface of attack for any potential threat actors. Ensuring that software updates are pushed as soon as we can ensure that they will not break the software is important, as this will help reduce vulnerabilities in our systems.