

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 2.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc48387396)

[Table of Contents 2](#_Toc48387397)

[Document Revision History 2](#_Toc48387398)

[Executive Summary 3](#_Toc48387399)

[Design Constraints 3](#_Toc48387400)

[Domain Model 3](#_Toc48387401)

[Evaluation 5](#_Toc48387402)

[Recommendations 8](#_Toc48387403)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 3.0 | 08/15/2020 | Joshua Gauthier | Added Recommendations section |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room wants to create a web-based implementation of their Android-based game. This will involve using server-side and client-side technologies in order to have a working web-based application. These front-end and back-end technologies are crucial to being able to develop and maintain the Draw It or Lose It web application.

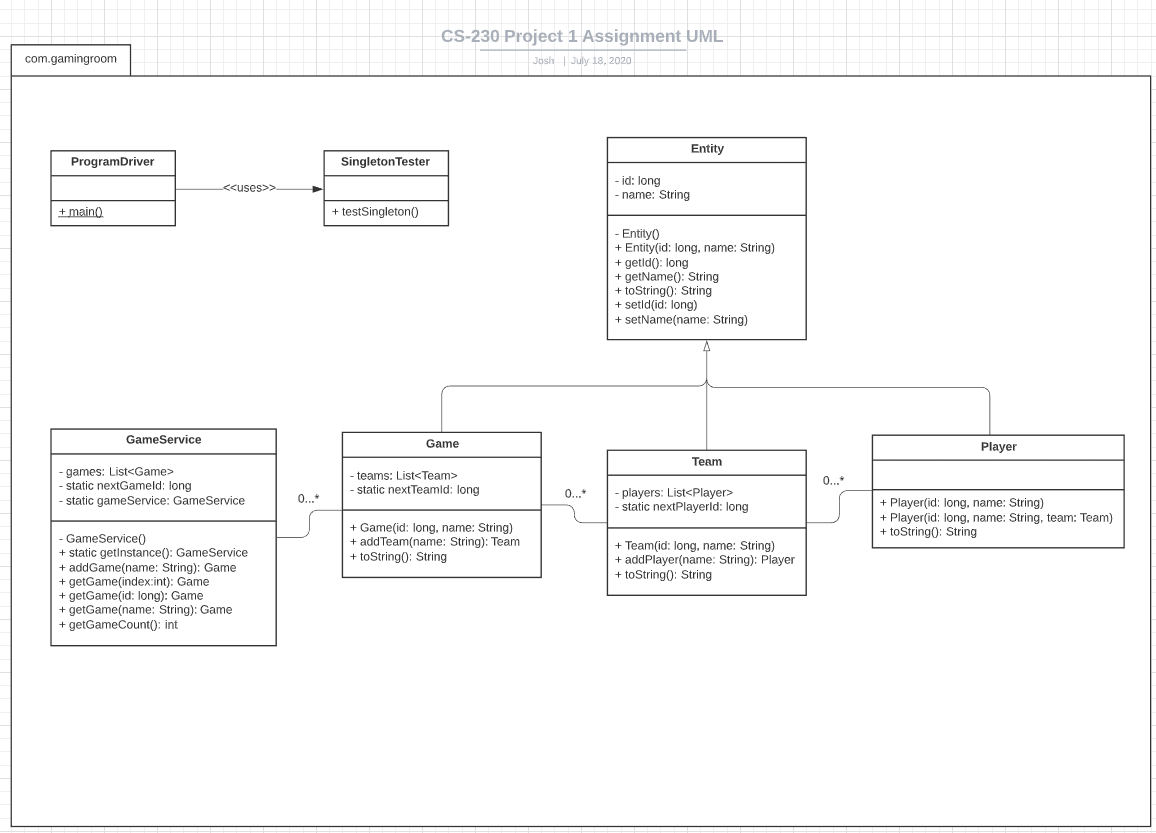
We will need server-side software that allows us to send web data to users’ machines. I propose running Tomcat on your windows cloud servers that you already have for the existing Android app. Tomcat is a web server software that works well with Java web applications. It is open source and free. We can develop the backend files we need in the Java programming language and they will work with Tomcat. We will also need a framework, so we don’t need to develop our code from scratch. Spring is the recommended framework for our purposes because it is popular for java web development and it is open source. We should use the same MySQL database that is in use for the existing Android app, so that the two apps are succinct. We will need to use the JDBC API to allow our Java program to work with the database.

Client-side technologies are needed in addition to our server-side choices, so customer devices can connect to our servers. When using CSS, we will need a framework that will help in writing the code, so we aren’t writing everything from scratch. Bootstrap is the most popular and supported. When developing the JavaScript code to make our web app more dynamic, we will need to utilize JavaScript frameworks and libraries. Angular is one of the most popular frameworks and React is a popular open source library that we can use.

## [Design Constraints](#_2et92p0)

* The game will be multiplayer, so people all over the world need to be able to log on to our server and request to join a game. This means our servers need to be able to handle the addition of all web-based players.
* The game is time-based, so this means our server and its technologies need to react quickly for users to experience no lag. The right server hardware is crucial here, but we also need to streamline our software design so that the program is efficiently built, which will allow the hardware to execute to its full potential. The same goes for front-end software, which also need to be streamlined so that clients aren’t a bottleneck.
* Someone on the mobile app should be able to play with someone on the web app. This means the web app and Android app should be cross-compatible. Our database should be the same for each app. The client should see the same GUI whether on Android or web.
* The web game will be available on traditional desktop operating systems using a modern browser. It will also be available to iOS users thru a native app. The web app must be responsive to varying screen sizes. Our front-end services must respond to various popular web browsers. Lastly, we need someone who knows HTML, CSS, JavaScript, Bootstrap, and Angular.

## [Domain Model](#_8h2ehzxfam4o)



Entity is the superclass and Game, Team, and Player are its subclasses. This inheritance relationship causes the subclasses to inherit Entity's methods and attributes. This makes the program design more efficient because there is not redundant code in each class. Instead of repeating id and name etc. within Game, Team, and Player, the data members are isolated into a parent class for modularity and efficiency.

Polymorphism is used in this UML design, as the toString() method in Entity is overridden in subclasses. This allows each subclass to inherit the toString() method while implementing it differently than the other subclasses. Attributes are all private, such as Entity.id and Entity.name. This acts as a security feature and fulfills the object-oriented principle of encapsulation. The attributes are accessed through mutator methods, which is good practice for OOP. This can help diminish side-effects and preventable bugs. If the main() method was able to directly access the objects’ attributes, the system would not be as secure, and it could be more difficult to say the program can reliably follow our software requirements.

There are list structures within GameService, Game, and Team. These lists allow multiplicity relationships between the classes. For instance, GameService has a zero to many relationship with Game. This essentially means there can be many Game objects in a GameService. Creating the classes with this type of multiplicity relationship allows us to meet the software requirements that games can have multiple teams and that teams can have multiple players.

SingletonTester is used by main() to verify that only one instance of the GameService can be constructed. It checks to make sure the singleton design pattern is functioning properly. This feature confirms the fulfillment of the software requirement “Only one instance of the game can exist in memory at any given time”.

## Evaluation

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | +macOS Server is cheap: $20  -macOS Server is meant for small businesses, not large-scale distributed environments  -Apple has been deprecating features of macOS Server for years | +UI is built outside of OS kernel, making it more stable than Windows  +Free and open source  +There are countless versions of this OS that can be used for distributed environment  +Most web servers are running some form of Linux, so there is a large community of support  -Must have expert to use it; may take time to learn | +Less time learning if already using Windows  +Well known and supported commercially  -Window Manager and Graphics Display Interface are within the kernel, making the OS less stable than macOS and Linux  -Expensive: at least $500 | +Android server apps are cheap: as low as $10  -Mobile devices are not meant to host scalable distributed environments and don’t have the processing power needed for our purposes |
| **Client Side** | -To develop and test the app on Safari and macOS, you need to own a Mac  -Macs cost at least $1000  -Need someone who knows the macOS ecosystem  -Must develop for Chrome, Firefox, and Safari  -The app must be responsive so the developers must know CSS and JavaScript  -Developers must have extensive knowledge of HTML | -It will not cost anything to test, as Linux is free  -Must have someone that knows the Linux OS to fully test the compatibility  -Must develop the app for Chrome and Firefox  -The app must be responsive so the developers must know CSS and JavaScript  -Developers must have extensive knowledge of HTML | -Must develop the app for Chrome, Firefox, and Edge  -Developers must have extensive knowledge of HTML  -The app must be responsive so the developers must know CSS and JavaScript  -Will not be very expensive to develop for and test compared to other platforms | -To develop for and test an iOS client, you need an iOS device such as an iPhone or iPad which will cost at least $300  -Need someone that understands the Android and iOS environments  -Need CSS and JavaScript code to be very responsive to different screen sizes and screen orientations  -It will take a good amount of time to mirror the existing Android app into an iOS app that is very responsive  -Developers must have extensive knowledge of HTML |
| **Development Tools** | -Java and SQL  -HTML, CSS, and JavaScript  -Eclipse, NetBeans, or IntelliJ IDEA for Java IDE  -Git, command line  -Tomcat for web server  -MySQL and JDBC API for database  -Bootstrap for CSS framework  -Angular JavaScript framework and React JavaScript library | -Java and SQL  -HTML, CSS, and JavaScript  -Eclipse, NetBeans, or IntelliJ IDEA for Java IDE  -Git, command line  -Tomcat for web server  -MySQL and JDBC API for database  -Bootstrap for CSS framework  -Angular JavaScript framework and React JavaScript library | -Java and SQL  -HTML, CSS, and JavaScript  -Eclipse, NetBeans, or IntelliJ IDEA for Java IDE  -Git, command line  -Tomcat for web server  -MySQL and JDBC API for database  -Bootstrap for CSS framework  -Angular JavaScript framework and React JavaScript library | -Java and SQL  -HTML, CSS, and JavaScript  -Eclipse or IntelliJ IDEA for Java IDE  -Git, command line  -Tomcat for web server  -MySQL and JDBC API for database  -Adobe PhoneGap for framework to develop mobile apps |

## Recommendations

1. **Operating Platform**:

Mobile operating systems are not designed for hosting distributed environments and macOS Server is only meant for hosting in small business networks. The two plausible OS choices for a large-scale distributed system are Windows and Linux. However, the recommended server platform would be a commercially released version of Linux. It is open source software that has proven stable through community collaboration, public testing, and commercial updates. As such, Linux has become flexible, powerful, and the most widely OS used for web servers. Linux is free, whereas it will cost $500 or more to run a Windows server OS. Because there is a large Linux community, there is always support for learning the operating system. For these reasons, Linux runs most modern web apps and thus hosts the majority of the internet.

1. **Operating Systems Architectures**:

Linux and Windows have different architectures because they come from different OS designs. Microsoft designed modern Windows OSs to have fast GUI responses. The Linux architectural design focuses on stability over fast UI response. Consider that kernel space has higher CPU and resource priority than user space, which makes processes in the kernel space faster. By placing the Window Manager and Graphics Display Interface in the kernel of the OS, Windows users experience less lag with the user interface. Linux places the UI in user space instead of kernel space, making the UI slower. However, when a problem happens on a UI process in Windows, the kernel crashes, causing the Blue Screen of Death. When a problem occurs in a UI process of Linux, that process is the only one that crashes. Thus, the Linux operating system is more stable because of its kernel and OS architecture.

1. **Storage Management**:

The server will need access to nonvolatile storage externally via a database. The most widely used database management system used with Linux is MySQL. It is an open source DBMS that also has paid options for extremely high-performing and scalable features. It is popular because it is fast, flexible, and has a large community of support. It is the recommended DBMS for the game app.

1. **Memory Management**:

Linux was developed from Unix which had an OS feature called driver memory isolation, or sandboxing. Linux inherits this feature that separates memory designation for drivers. Although Windows uses driver memory isolation, Microsoft favors easy system extensibility over maximum security. Thus, Windows is more vulnerable to protection and security threats. The driver “sandboxing” inherent to Linux makes the OS much more stable. What this means is the Linux web server will crash less and be more reliable while hosting Draw It or Lose It.

1. **Distributed Systems and Networks**:

Draw It or Lose It will be a game that utilizes the client-server architectural pattern. The web application will be hosted by a server (or servers) that will provide perpetual connectivity through the internet. Customers will use client platforms to send a request to the REST API on the server. This uniform resource locator (URL) request sent to the RESTful API is an HTTP protocol data transfer that asks for a resource from the server. The API looks at the URL’s directory specifics, then may execute a specified API method with or without parameters. The web server then responds to the client with a representation of the requested resource. This response can be in JSON format. The client then parses the JSON response and the resource representation is displayed to the end user. This client-server architecture will allow the required 1000 clients sending requests concurrently. The REST-style API will allow flexible and efficient distribution of application services and will be necessary for handling the large amounts of incoming and outbound transmissions. The native iOS and Android apps will be designed to work with the RESTful API and JSON responses, aiding the cross-platform distributed design. To be prepared for periodic server maintenance, more than one server can be implemented so there is less web app downtime. To prepare for natural disasters, several servers can be placed in different geographical locations. The Game Room’s LAN can have a demilitarized zone that holds the web servers, while the database can sit within the private network.

1. **Security**:

The API will first authenticate a client by asking for a username and password. The client’s credentials are checked against a MySQL database behind a firewall. If the credentials authenticate successfully, the API will grant specific authorized privileges to the user. An access token will be granted to the client, which is then needed at the header of each subsequent data transfer from that client. The Linux server is substantially secure against protection and security threats because of its extensive driver memory isolation. This acts as an additional shield against possible runaway processes trying to gain access to OS objects and domains that they don’t have authorization for. Browser and mobile OS security are understood by the development team as to integrate the API with clients securely. The web servers will reside in a demilitarized zone behind a firewall, and the private network holding the users’ sensitive information will reside behind another firewall in a private network.