

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## 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)4

[**Evaluation**](#_2o15spng8stw)5

[**Evaluation**](#_2o15spng8stw)**(cont.)** 6

[**Evaluation**](#_2o15spng8stw)**(cont.)** 7

[**Recommendations**](#_m8aleynsvzvc)8

[**Recommendations**](#_m8aleynsvzvc)**(cont.)** 9

## 

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/18/2021 | Uriah Fadum | Created Software Design Documentation  Completed Executive Summary  Completed Design Constraints & Tech Requirements  Completed System Architecture View  Completed Explanation of Domain Model |
| 1.0 | 04/03/2021 | Uriah Fadum | Started Development Requirements Table |
| 1.0 | 04/04/2021 | Uriah Fadum | Completed Development Requirements Table |
| 1.0 | 04/14/2021 | Uriah Fadum | Started Requirements Section |
| 1.0 | 04/18/2021 | Uriah Fadum | Completed Requirements Section |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## [Executive Summary](#_sbfa50wo7nsh)

Gaming Company “The Gaming Room” operates an android OS application interface, called “Draw It or Lose It”. In order to effectively target a broader base of users/devices, they have decided to develop a web-based version of the application. Several key issues will need to be addressed starting with reviewing existing android application server/services, application architecture and existing client-server design (including programming languages used). After reviewing current application operations, careful consideration will determine the appropriate integration of both the original Android App and the new Web-Based version of the game. Considering the likelihood of the original android application being written in either C, C++, JavaScript or Java, our plan to move forward includes using Java as the main language used to create the web-based version of “Draw It or Lose It”.

## [Design Constraints](#_2et92p0) & Software Requirements

Web-Based Interface (Multi-OS Architecture Compatible)

Game(s) have the ability to host multiple teams

Team(s) have the ability to host multiple players

Unique Game and Team names to verify gaming service/name not already in use

Dev-Op Personnel (Current & Proposed)

Web Server/Application Hosting Services (Current & Proposed)

Programming Language(s) (Current & Proposed)

Web/Application Security (Current & Proposed)

Server Security (Current & Proposed)

Client Security (Current & Proposed)

User Integration (Account)

Legal/Licensing

Marketing/Advertising

Testing/Beta-Testing

## [System Architecture View](#_ilbxbyevv6b6)

Some considerations would be to implement a n-Tier or multi-tier architecture. This would then address issues of scalability, bandwidth and system control that would be faced when designing an application and implementing a strategy for integration and security.

## [Domain Model](#_8h2ehzxfam4o)

The java package com.gamingroom contains the source code for creating and initiating unique games with unique teams that are made up of one or more players. Within the code is the ability to control the creation of game specific service with the singleton design pattern, thus the singleton tester class. Also within the application is the program driver class with the ability to initiate and test all other package classes. The other four classes are considered the working parts of the actual package. The Entity class is the base/super class of game, team and player classes and follows industry standards and best practices by implementing inheritance and polymorphism when creating game, team and player objects. The game service class is the game engine with the ability to create and initiate unique games.

****

## 

## 

| **Evaluation** | | | | |
| --- | --- | --- | --- | --- |
| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| **Server Side** | **Advantages**  -MacOS X Server is User-friendly when used w/ ONLY Apple products.  -Quality server support and products.  -Does NOT use Active Directory natively but can be connected to a Windows server/network using AD via LADPv3.  - Higher security due to the use of proprietary technology.  **Disadvantages**  -NOT easily integrated w/ other platforms without in-depth experience in MacOS X Server management.  -High initial hardware/software cost to initialize and host a website application.  -High cost to scale up, due to equipment costs. | **Advantages**  -Easily able to deploy, scale-up & manage a web application server/service.  -Most software is free and regularly updated.  -Low hardware & software cost to initialize and host a website-based application.  -Does NOT use Active Directory natively but can be connected to a Windows server/network using AD via LADPv3.  -Easily able to incorporate cloud-based features & services.  -Can operate on most computing equipment.  -Highly adaptable to computing environments and systems.  **Disadvantages**  -Large attack vector for security vulnerabilities.  -Expert knowledge required to create, operate and manage server operations. | **Advantages**  -Easily able to deploy, scale-up & manage a web application server.  -Quality server support and products.  -Easily able to incorporate cloud-based features & services.  -Uses Active Directory Protocols natively.  -Widely used server environment.  **Disadvantages**  -Large attack vector for security vulnerabilities.  -Expert knowledge required to create, operate and manage server operations. | **Advantages**  -Mobile…  **Disadvantages**  -expensive to program, deploy and maintain.  -Does NOT use Active Directory or LDAP natively and would require additional software to accommodate AD & LDAP use.  -Cap on scalability.  -Cap on Bandwidth.  -Cap on Application size. |
| **Client Side** | -Designing a web-based application that would utilize native safari browser features (Desktop & Mobile) would be top priority when assessing the MacOS X & iOS platforms.  -Would require expertise in Swift, Objective-C and other OS related languages. | -Designing a web-based application that would utilize Firefox browser features (Desktop & Mobile) would be top priority when assessing the client side of different Linux platforms.  -Would require expertise in programming in C, C++, Java and other mainstream programming languages (i.e., Python, Perl, etc.). | -Designing a web-based application that would utilize the native Microsoft Edge browser features (Desktop & Mobile) would be top priority when assessing the client side of the different WIN32, 64 & Windows 10 mobile platforms.  -Would require expertise in programming in C, C++, Java and other mainstream programming languages.  -Does have Licensing costs when using Microsoft features & services. | -Designing a web-based application that would utilize the native Android mobile browser features would be top priority when assessing the client side of the different mobile platforms.  This would have to be addressed & researched since some mobile device manufactures produce different variants of browser applications with their products (i.e., Huawei browser running on Harmony OS 2.0+, KaiOS,  etc.)  -Would require expertise in programming in C, C++, Java as well as a comprehensive background in Mobile OS architecture and developmental tools used. |
| **Development Tools** | Most modern web browser technology has been programmed at the core using C++ for developing purposes, utilizing other languages to best suit the need of whatever web function is being addressed and whatever core OS is being used.  -MacOS X & iOS use C, Objective-C and others like Swift and JavaScript + corresponding IDE’s. | -Linux systems utilize a majority of the mainstream programming languages and IDE’s (i.e., Eclipse, etc.)  -offers the ability to effectively program in every language.  -Linux has a large collection of corresponding program language libraries & tools.  -Excellent cross-platform programming capabilities. | -Win32 & 64 based platforms use C# and C++. Other languages are used such as Java, HTML and others and for the purposes of a webpage-based game on Microsoft Edge, these would be the main programming languages used, plus corresponding IDE’s. | -Most mobile applications are based on Android OS and are developed using Java, C, C++, HTML5 and the use of Android Studio. If not, then they are proprietary to their corresponding manufacturer and can use whatever language IDE the manufacture of the device uses (i.e., iOS=Xcode, Linux Mobile=Xaramin, etc.). |

## Recommendations

1. **Operating Platform**: Selecting the Linux Operating System will allow for maximum system flexibility & control of both the software development as well as the selection of server/mainframe hardware. Since Linux is highly adaptable for operation on most computational equipment, utilizing the Linux kernel addresses issues like, scalability, cross-platform application development & deployment, cloud-migration & development as well as security and system management. Since Linux is also open-source, developers can tailor make a minimal OS platform to maximize the hardware capabilities of the server system. In turn, discarding un-used OS applications and OS features that would otherwise bloat the operating system and consume vital server resources. Since both Microsoft Windows & Apple use a proprietary server operating system (Windows Server 2019™, MacOS Server™), this would mean that some amount of licensing fees for use would incur if one of these OS’s are used. It would also mean that a large dependency on proprietary technology would have to be researched and addressed, making using Linux a cost-effective workaround for development. Using Linux not only cuts the licensing fee’s out but also puts the control of the system back into the hands of the company and respective system developers, allowing for a system architecture tailored to maximize all application related operations.
2. **Operating Systems Architectures**: Since Linux is a widely adopted server OS, creating a multi-tier web application architecture using the Linux kernel could prove to be highly efficient. Structuring the application into several smaller functional parts (Microservices) and creating a minimal OS to host the applications’ services/functions could prove to be the most efficient overall. This then would mean that the architecture of the program application would be a multi-layered system with a majority of the functionality in the application layer (web app) and using the Linux OS & relevant application programming for the server-side communication & database storage of the multi-layered application and its respective user base. Separating the application into smaller microservices also provides the application an opportunity to incorporate different areas to handle security, caching, app presentation, networking services, etc. Aligning the applications structure like the structure of most cloud-based architectures proves to solve most of the concerns of the client-server communication problem as well as providing a highly efficient and standardized approach to a network-distributed application architecture.
3. **Storage Management**: Creating a separate storage database layer within the application/mainframe architecture could prove to be beneficial in a few different ways. One, more control of the system and its overall security can be addressed by separating the layers of the application to include an encrypted storage area. Two, it allows for enhanced communication, since the layer itself could be migrated into a service itself and then streamlined for network deployment. Three, it continues to align to a non-monolithic application architecture, in turn, making it easily migratable to main-stream hosting service providers such as AWS, Azure, IBM Cloud, etc., if application development & user volume exceeds expectation.
4. **Memory Management**: Linux distributions, like other mainstream Operating Systems (i.e., Windows, Apple and Mobile) utilize Random Access Memory. The main difference is how the RAM is being managed by the system. Linux creates a virtual memory layer and assigns RAM in accordance with a demand paging system. Combining Linux’s efficient use of RAM and a hardware selection that factors storage scalability & RAM capacity. Linux would effectively handle application memory needs under any and all circumstances.
5. **Distributed Systems and Networks**: When it comes to the communication of relevant gaming application data, Linux offers plenty of options for handling the needs of the Draw It or Lose It application. Since a majority of the application is developed using Java and other open-source APIs like Apache Maven, etc., Publishing a close-to-real-time gaming application would depend on the functionality of the server software. (e.g., Performance related concerns, i.e., Apache vs. NGINX), Bandwidth of ISP, and server hardware. Depending on what level of internet bandwidth the company will want to operate on, will directly reflect on the hardware capabilities and vice versa. In hindsight, Linux offers the ability to make necessary changes in case application development or network demands exceeds expectation.
6. **Security**: When it comes to security, Linux offers a wide range of standard OS security features like, user/group privilege management, system & application auditing and as well as other features used by hardware manufactures, i.e., TPM, HPM, etc. This coupled with appropriate application security programming, will be able to handle the relevant security concerns of the gaming application, Draw It or Lose It.