

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 |
| --- | --- | --- | --- |
| 3.0 | 08/12/20 | Jef DeWitt | Changes in this revision include updates to the following section: Recommendations. |

**Instructions**

Fill in all bracketed information on page one (the cover page), in the Document Revision History table, and below each header. Under each header, remove the bracketed prompt and write your own paragraph response covering the indicated information.

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room currently has an Android-only game, Draw It or Lose It. They are interested in developing web-based versions of this game that serves multiple platforms. Development environments will need to be set up for Mac, Linux, Windows, and mobile devices. This project will use client-server architecture. One specific design pattern to note, only single games can be played at a time. The singleton design pattern will be used to prevent multiple games from existing in memory at any given time. As for gameplay, games allow for one to many teams to be involved, teams can have one to many players, and game and team names must be unique.

## [Design Constraints](#_2et92p0)

* Game versions are run in-browser, so we are constrained by browser features and performance
* Targeted browsers are Chrome, Firefox, Safari, and Edge. Supported versions need to be specified to determine cutoff point as to maintain best experience for most users.
* Gameplay may be affected by responsive layouts that accommodate large (desktop), medium (laptop), and mobile devices (tablets and phones).
* Data and logic layers must be failure-tolerant. Therefore, a three-tier distributed system (database, web-server, client) will be used where database and web-server layers will be cloud-based (AWS). Server instances to be spun up as needed.
* Data requests must use HTTP. Logic layer performance must be prioritized so as not to negatively affect gameplay. This also includes limiting the size of data payloads returned from the backend.

## [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)

An important class to note is the *Entity* class; encapsulating related properties and methods.This class will act as the superclass from which *Game, Team,* and *Player* inherit. This means that these classes will share a common list of attributes and methods. The *toString()* methods of each subclass are great examples of polymorphism; they override the original method of the same name in the Entity class.

The *GameService* class can create 0 to many instances of a game; bearing in mind, only one game exists in memory at a time. It follows that the *Game* class instantiates one to many teams and the *Team* class instantiates 0 to many players per team.

Lastly, the *ProgramDriver* class will act as the driver of the application, kicking off program execution and instantiating other classes. Note, *ProgramDriver* uses the *SingletonTester* class to enforce the singleton design patter mentioned above.

****

## 

## [Evaluation](#_2o15spng8stw) (see next page)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Characteristics:  Mac hosted website will run on any operating system  Advantages:  Mac hosting environments are user friendly  High security  Integration with Xcode server, Jenkins, aids in continuous integration  Weaknesses:  Hosts are hard to find, rare  Cost more | Characteristics:  Linux hosted website will run on any operating system  Advantages:  Lowest cost option  High security  Open-source, highly configurable  Weaknesses:  Requires more expertise to use than Mac or Windows | Characteristics:  Windows hosted website will run on any operating system  Advantages:  Popularity makes it easier to find programmers  Weaknesses:  Moderate security  Similar price tier with Mac  Require frequent rebooting  High volume sites may run poorly | Characteristics:  Mac, Linux, and Windows hosted sites will run on mobile browsers  No server necessary for native apps  Upload iOS apps to App Store and Android apps to Play Store  Advantages:  No hosting necessary for native apps  Weaknesses:  App store has rigorous approval process  App Store has annual fee plus fees from app earnings  Play Store has one-time fee plus fees from app earnings |
| **Client Side** | Developer cost:  Moderate as most devs have experience with the clients listed below.  Development time:  Quick to average as game already exists and will only need converting to these new platforms.  Developer expertise level:  Mid-level  Clients (Browsers):  Safari (WebKit)  Chrome (Blink)  Firefox (Gecko) | Developer cost:  Moderate as most devs will have experience with the clients listed below.  Development time:  Quick to average as game already exists and will only need converting to these new platforms.  Developer expertise level:  Mid-level  Clients (Browsers):  Edge (Chromium)  Chrome (Blink)  Firefox (Gecko) | Developer cost:  Moderate as most devs will have experience with the clients listed below.  Development time:  Quick to average as game already exists and will only need converting to these new platforms.  Developer expertise level:  Mid-level  Clients (Browsers):  Edge (Chromium)  Chrome (Blink)  Firefox (Gecko) | Developer cost:  Moderate to high, bearing in mind that native app devs are more niche than web devs  Development time:  Average to long, a web-based app will get developed faster than a native app. iOS apps have more hurdles to pass thru to reach market, potentially slowing down product  Developer expertise level:  Mid-level to High, mobile web-based vs native app  Support needed for various iPhone versions.  Clients (Browsers):  Safari (WebKit)  Chrome (WebKit) |
| **Development Tools** | Frontend languages:  HTML  CSS  JavaScript  Backend languages:  PHP  Java  Database:  SQL  IDEs:  Visual Studio Code  VSCode  Atom  WebStorm (License cost)  IntelliJ (License cost)  Mobile Development Considerations  Use browser Developer Tools for responsive layout development (i.e., desktop, laptop, tablet, and phone window sizes) | Frontend languages:  HTML  CSS  JavaScript  Backend languages:  PHP  Java  Database:  SQL  IDEs:  Visual Studio Code  VSCode  Atom  WebStorm (License cost)  IntelliJ (License cost)  Mobile Development Considerations  Use browser Developer Tools for responsive layout development (i.e., desktop, laptop, tablet, and phone window sizes) | Frontend languages:  HTML  CSS  JavaScript  Backend languages:  PHP  Java  Database:  SQL  IDEs:  Visual Studio Code  VSCode  Atom  WebStorm (License cost)  IntelliJ (License cost)  Mobile Development Considerations  Use browser Developer Tools for responsive layout development (i.e., desktop, laptop, tablet, and phone window sizes) | Development Considerations:  Mobile development occurs on non-mobile platform (Xcode requires Mac for native iOS development). Thus, desktop/laptop machines will be necessary for development  Languages:  Swift  Objective-C  JavaScript (using hybrid app framework like Ionic or for web app)  IDEs:  Xcode (Swift/Obj-C)  Visual Studio Code  VSCode  Atom  WebStorm |

## 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**:

For the web-based version of the Draw It or Lose It app, I recommend Linux servers for web hosting due to their low cost and high level of security. Proper configuration/setup may take a bit longer but it is worth the time investment. Amazon Web Services (AWS) provides cloud computing options that meet our needs. These Linux distributions will be hosted on AWS Elastic Compute Cloud (EC2) virtual server instances.

1. **Operating Systems Architectures**:

Targeted browsers are listed below allowing for current version support. Support for legacy versions will be determined based on market share but generally will not cover browsers more than two years old from present date.

* Safari (WebKit)
* Chrome (Blink)
* Chrome (WebKit)
* Firefox (Gecko)
* Edge (Chromium)
* iOS (App Store)

1. **Storage Management**:

AWS is recommended for cloud storage as it is much more cost effective, paying for the amount of storage used, versus purchasing hardware that constantly needs to be provisioned, maintained, and upgraded year after year. We can meet our cloud storage needs using Amazon Simple Storage Service (S3). Our database type will be PostgreSQL as it is open-source with years of support behind it.

1. **Memory Management**:

Several recommendations to ensure efficient utilization of memory resources:

1. Sizing game images correctly to their container size
2. Using the appropriate format; JPG is recommended
3. Using image compression to limit size
4. Use proper network request headers to enable image caching where appropriate

Note, browser memory performance considerations are shown below:

* Safari (WebKit) – fastest web browser, using least RAM
* Chrome (Blink) – heavy RAM usage, most of any browser
* Chrome (WebKit) – WebKit implementation uses less RAM than Blink (mobile, iOS)
* Firefox (Gecko) – third most RAM consumption
* Edge (Chromium) – second most RAM usage behind Chrome (Blink)
* iOS – Automatic Reference Counting (ARC), non-browser dependent

1. **Distributed Systems and Networks**:

We are going to use a three-tier distributed system consisting of:

* PostgreSQL databases
* Middle-tier of Linux-based virtual server instances
* Clients (mobile/desktop browsers and native apps)

These tiers will be cloud-managed, meaning that new instances of servers can be “spun up” as needed when workload increases, such as when a viral social media campaign increases game adoption. This has the added benefit of being outage-resistant as there isn’t a central concentration of hardware that, if negatively affected by a power outage, for example, could disrupt gameplay. Bearing this in mind, these systems are not completely failure-proof, and independent failures and nondeterminism (not knowing which node failed) are two issues that do plague distributed systems.

Connectivity on the AWS side of things can be securely managed via VPN, AWS Direct Connect, AWS Direct Connect + VPN, and AWS VPN CloudHub. Obviously, as a web-based application, game performance is largely dependent on the end-user’s network speed. Without high-speed network connections, gameplay will be adversely affected.

1. **Security**:

From a backend and database perspective, hosting on Linux machines affords a high level of security. AWS allows manual security tasks to be automated and is a reputable industry leader with many clients that have top-secret workloads.

Authentication and authorization will be required for making changes to our stack in AWS. This strategy of authenticating users and allowing role-based access is also recommended on the client-side. On the frontend, these user roles limit who can edit game settings, allowing control only to team captains’, for example.

As far as native mobile devices are concerned, iOS comes with Data Protection, a built-in encryption/decryption feature that is automatic and hardware accelerated for storing files in the app directory.

Note, browser security comparisons are shown below:

* Safari
  + The good:
    - Prevents suspicious sites from loading and alerts to potential danger
  + The bad:
    - Not open-source, updates at irregular intervals
* Chrome
  + The good:
    - Largest market share at almost 80%
    - Auto updates every six-to-eight weeks
  + The bad:
    - Google tracks everything to monetize information
    - Close-source, no way to see what is hidden in the code
* Firefox
  + The good:
    - Best, latest security features
    - Allows “Content Blocking” to block trackers that the browser detects
    - Completely open-source
  + The bad:
    - Doesn’t update as frequently as Edge or Chrome but still at regular intervals
* Edge
  + The good:
    - Now Chromium-based, open-source, and auto updates at least once a week
  + The bad:
    - IP addresses can be linked to backend servers via identifiers