

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)3

[**Evaluation**](#_2o15spng8stw)4

[**Recommendations**](#_m8aleynsvzvc)6

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 09/16/2020 | Kyle Hake | Constructed Executive Summary, Design Constraints, and Domain Model. |

**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)

Problem Statement: The client, The Gaming Room, wants to develop a web-based game that is available on multiple platforms from their current game, Draw It or Lose It, which currently is only available on Android. The development for the web-based version is being outsource to our team to help streamline the process.

Software Requirements:

* A game will have the ability to have one or more teams
* Each team will have players assigned to it
* Game and team names must be unique, and checks whether it is unique when choosing a name
* Only one instance of the game can exist in memory at a given time, accomplished by unique game, team, and player names.

## [Design Constraints](#_2et92p0)

1. Application Architecture – Currently the app is exclusively used on Android, so the security settings will need to be adjusted for multiple different operating systems. Web hosting of the application will also have to be considered, and allocation of resources to scale based on a client – server architecture pattern needs to be evaluated as well. Using a RESTful API will also be important so that different operating systems can interact with the server seamlessly to provide the user with the best experience.
2. Storage and Memory – Since this web based application will be using a client – server architecture, the storage and memory requirements for the app need to be considered when developing the game. For example, the app could download all the images/media with the initial download, but it would take up more space on the device that is running the app. On the other hand, the media could stay on the server side but would require the device to pull data from the server to load the images as the game is running. The first situation would be ideal for a device with ample memory, and the latter would be ideal for older devices.

## [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)

Below is the UML class diagram of the project discussed above. The ProgramDriver class contains the main() function of this project and uses the SingletonTester to test the singleton design pattern used in GameService. GameService has a singleton pattern design, which means its default constructor is private. The GameService can have 0 to many games, but only one instance of a game based on the singleton pattern. The Entity parent class to the Game, Team, and Player classes. Each game can have zero to many teams as well as each team having zero to many players.

OOP principles are used to satisfy different software requirements presented by the client. Through abstraction, a game can have multiple teams by using the addTeam method, taking an object of the Team class and adding it to the teams list of the Game class. Each team can have players assigned to it using the addPlayer method in the Team class, which functions similarly to the addTeam method. Variables are encapsulated by being private, and the classes use getters to keep the information protected. The Entity base class is inherited by the Game, Team, and Player classes. This is used to hold common attributes, which can be used to check if a name of a Game or Team is unique. Polymorphism is used in both the Entity and GameService classes. GameService can get an instance of the Game class by using its id or name, and an entity can be constructed publicly with an id and name or privately as default. These can be used to make sure that one instance of the game is allocated to memory at a given time, using unique game, team, and player names.

**Diagram

Description automatically generated**

## [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** | **Operating System Structure:** Closed source, Licensed  **User Admin:** Specific type of user account has to be assigned. Can make other user accounts, install software, and adjust security  **LDAP?:** capable but not native  **Docker/VM:** Both  **Server Implementation:** Through new App available on app store  **Cloud:** No cloud OS | **Operating System Structure:** Open Source, Not licensed  **User admin:** Specific account, other users can use ‘sudo’ to run with privileges of superuser.  **LDAP:** Not installed, easy implementation  **Docker/VM:** Both  **Server:** Many different options, all console line based, open sourced.  **Cloud:** Yes | **Operating System Structure:** Closed Sourced, licensed  **User Admin:** Specific type of user account, has added controls over security  **LDAP:** Installed  **Docker/VM:** Both  **Server:** Closed sourced software, now available on Azure cloud  **Cloud:** Yes, Azure | **Operating System Structure:** Various, depends on device.  **User Admin:** Mobile devices are single user, has authentication to get into the device itself.  **LDAP:** capable  **Docker/VM:** Available on both apple and android  **Server:** Not possible  **Cloud:** |
| **Client Side** | **IE:** Discontinued  **MS Edge:** Works well on Mac products, available via app  **Chrome:** Available via download  **Firefox:** Available via download  **Safari:** Native to Mac  Safari is best for mac because of its connection to the Apple ecosystem and focus on privacy and security | **IE:** Not capable  **MS Edge:** Not available yet  **Chrome:** Available via download  **Firefox:** Native to Linux  **Safari:** Native to Mac  Google chrome is best for Linux because it outperforms Firefox | **IE:** Native but outpaced by Edge  **MS Edge:** Native to windows  **Chrome:** Available via download  **Firefox:** Available via download  **Safari:** Not available on windows  MS Edge is the superior to other browsers | All browsers are available for the various types of mobile devices.  IPhone: Safari tops out Chrome because of its Apple ecosystem integration and privacy.  Android: Chrome would arguably beat out other browsers for speed and ease of development. Only con would be size of app. |
| **Development Tools** | Programming Languages:  Frontend – Javascript  Backend – Python, Ruby  Databases – SQL  IDEs – Jetbrains products (licensed)  VSC – JS Python – IDLE, PyCharm (public)  Ruby – VIM, Atom  SQL – Squirrel, TablePlus | Programming Languages:  Frontend – Javascript  Backend – Python, Ruby  Databases – SQL  IDEs – Jetbrains products (licensed)  VSC – JS Python – IDLE, PyCharm (public)  Ruby – VIM, Atom  SQL - Squirrel | Programming Languages:  Frontend – Javascript  Backend – Python, Ruby  Databases – SQL  IDEs – Jetbrains products (licensed)  VSC – JS Python – IDLE, PyCharm (public)  Ruby – VIM, Atom  SQL - Squirrel | Development isn’t extensively done on mobile devices, desktops are preferred. |

## 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**: Mac OS has an easy to implement server management application that is available from the App Store. Mac OS has similar security measures in terms of user authentication and admin privileges that other operating systems have, is LDAP capable with Open LDAP, and easy to use virtual machines as well as Docker functionality. Then only drawback to Mac OS is its lack of a cloud OS, but with a project like this, isn’t a necessary feature.
2. **Operating Systems Architectures**: Mac OS’s kernel is open - sourced and is free for anyone that is using various Apple devices. The OS itself is a hybrid structure that has a layered structure for the application environments, interface, and graphics, and a microkernel layer with the BSD kernel (Darwin). Inside Darwin, Mach is in charge of data flow, memory use, and fundamental tasks of the processor. The graphics layer has separate components that are in charge of 2D and 3D graphics as support for different types of digital media. The application layer has different components that support classic applications, porting existing applications to a standard API, as well as OOP environments and Java capabilities. The final layer is the UI.
3. **Storage Management**: Storage management in terms of any application or software are going to be pretty similar on any of the various platforms. An inexpensive solution to data storage would be setting up a cloud for back-ups of the application as well as data that would be considered archived or not accessed regularly. This is because moving data to and from a cloud can take more time than physical storage such as an SSD. SSDs would be a viable option for data that needs to be readily accessible for application function or maintenance. Also, having a local database for storage for server outages would be a good idea as well.
4. **Memory Management**: Mac OS uses a memory management system that will keep an application on the RAM even after its closed and marks it as “inactive memory”. Once the RAM is full, the operating system can swap out new processes for those in the “inactive memory” bank to run. This allows regularly access applications to run faster because it doesn’t haven’t be pulled from the hard drive, but still allows processes to be switched out if needed.
5. **Distributed Systems and Networks**: The client – server model, which is implemented for this application, will primarily rely on the internet to provide the connectivity needed for intercommunication between various users and the server itself. With that being said, important things to consider when using this type of architecture are server congestion, and system outages. To make sure the launch of the app is successful, pulling information from the current mobile app would be a good place to start to evaluate the possible traffic of the app. This would be important to figure out how many users the server would need to handle at a given time and provide the benchmark of testing to make sure the server can handle the traffic before going live. Also, having someone to manage the server would be an added cost, but would be essential in preventing outages and getting the server back online in case of an outage.
6. **Security**: Security is an important component of the software development process because keeping user information safe is paramount especially if sensitive data is collected for making a profile in the game. To secure a client – server model, firewalls and network segmentation can be used to protect clients form the server and vice versa. Also, when developing the application, evaluating for weak points that could compromise the app and give way to malicious users will also be important before making available to the public. Information such as name and date of birth are common pieces of data collected for web - based games, so keeping this data safe is important for user trust in the product. Using end to end encryption as well as encryption on local devices will help protect personal identifiable information. Mac OS is considered one of the most secure operating systems out there with its built – in encryption services as well as biometric touch ID available on newer models. Its web browser, Safari, has built in privacy features that out-pace other browsers and warns users when accessing potentially harmful sites that could lead to data being accessed by outsiders.