

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 | 04/16/22 | Blake Condrey | Software Design Prototype |

**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 has a popular Android app titled “Draw It or Lose It”, and would like to expand on the success by developing a web-based version of the game to serves multiple platforms of the successful app. The game should have the ability to have on or more teams involved, with each team have multiple players assigned to the respective team. Games and team names must be unique to allow user to verify if a name is in use, and only one instance of a game can exist in memory at any time. The Gaming Room staff does not know how to set up the environment, and has requested the assistance of CTS to facilitate streamlining the development.

## [Design Constraints](#_2et92p0)

Given the game currently only exists on Android platforms, a constraint exists in migrating the application to at least two separate platforms, specifically iOS and the web, which have unique operating systems.

The program to serve the game must be outfitted to meet requirements for successful deployment on three operating platforms.

The program must be able to allow one or more concurrent teams to be created, agnostic of users’ respective operating platform.

Notification to team creator, regardless of platform, must be rendered if a team name already exists within the game instance.

Creation of unique identifiers for a game, teams, and players, to ensure that only one instance of a game exists in memory.

## [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)

Beginning with the ProgramDriver class, which contains the entry point – the main() method - a direct association is in place with a SingletonTester class.

The SingletonTester employs the Singleton pattern to validate whether an instance of GameService exists.

The GameService class contains list of Games with the ability to establish new games, and their respective team and player unique identifiers, while employing the iterator pattern to ensure that a unique instance of a game already exists. Lastly, the GameService class exhibits multiplicity, a logical association of zero-to- many relationships with the Game class.

The Entity class serves as the parent class to the child classes of Game, Team, and Player, which all inherit the attributes of the Entity Class.

The Game class extends from Entity, following the inheritance relationship, which has a list of teams for the unique game, and can establish a new Team while verifying that a unique instance of a Team already exists by employing the iterator pattern. Lastly, the Game class exhibits multiplicity, a logical association of zero-to-many relationships with the Team class.

The Team class extends from Entity, following the inheritance relationship, has a list of players, and the ability to establish new players, while verifying that a unique instance of a player already exists by employing the iterator pattern. Lastly, the Team class exhibits multiplicity, a logical association of zero-to-many relationships with the Player class.

The Player class extends Entity, following the inheritance relationship, and constructs the player’s unique identifier and name.

**"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)

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** | **Characteristics:**  A web app hosted on MacOS has the capability of running on any OS.  **Advantages:**  MacOS has the advantage of a user-friendly environment.  Out-of-the-box tight security features.  Continuous Integration / Continuous Delivery through Xcode.  **Weaknesses:**  Host availability is rather small.  Not as cost-efficient as other OS. | **Characteristics:**  A web app hosted on Linux has the capability of running on any OS.  **Advantages:**  As far as cost goes, Linux is the most cost-efficient.  Open-source OS with several security experts contributing to OS security. Open-source characteristic provides options for custom configuration.  **Weaknesses:**  Due to open-source, will require some higher expertise or experience. | **Characteristics:**  A web app hosted on Windows has the capability of running on any OS.  **Advantages:**  Wide-scale use of Windows has subsequently created a massive pool of adept developers.  **Weaknesses:**  High price-points.  Latency with high-volume applications. | **Characteristics:**  A web app hosted on MacOS, Linux, and Windows has the capability of running on mobile platforms.  Apps can be featured on App Store and Play store for respective OS, giving wide visibility.  **Advantages:**  Mobile-native apps do not require hosting.  **Weaknesses:**  App Store is costly; fees incurred on annual basis, as well as 15% cut from app’s earnings. |
| **Client Side** | Developer cost:  Moderate-to-high.  Developers should have experience with browsers (see *Clients* below.  Development time:  Tech-debt should be low. The App exists and only requires migration steps.  Developer expertise level:  Mid-level career experience.  Clients:  Safari  Chrome  Firefox | Developer cost:  Moderate-to-high.  Developers should have experience with browsers (see *Clients* below.  Development time:  Tech-debt should be low. The App exists and only requires migration steps.  Developer expertise level:  Mid-level career experience.  Clients:  Edge  Chrome  Firefox | Developer cost:  Moderate-to-high.  Developers should have experience with browsers (see *Clients* below.  Development time:  Tech-debt should be low. The App exists and only requires migration steps.  Developer expertise level:  Mid-level career experience.  Clients:  Edge  Chrome  Firefox | Developer cost:  Moderate-to-high.  Mobile Developers should me mobile-first developers, as mobile development is a unique category in development.  Developers should have experience with browsers (see *Clients* below.  Development time:  Tech-debt is higher than average. Mobile app development will have rigorous approval process to reach respective app stores. Release to market is expected to be longer than web release.  Developer expertise level:  Mid-level career experience, to high experience and expertise.  Clients:  Chrome  Safari |
| **Development Tools** | **Frontend languages/Frameworks:**  HTML  CSS  JavaScript (React.js, Angular, Vue.js are popular and common frameworks.)  **Backend languages:**  Java  Node.js  **Database:**  SQL databases.  (MySQL, PostgreSQL, SQLite)  **IDEs**:  VSCode (Most popular, doesn’t behave well with Java)  Visual Studio  NetBeans  Eclipse  **Responsive Design:**  In development, dev teams can inspect product through developer tools to test breakpoints and responsiveness for CSS. | **Frontend languages/Frameworks:**  HTML  CSS  JavaScript (React.js, Angular, Vue.js are popular and common frameworks.)  **Backend languages:**  Java  Node.js  **Database:**  SQL databases.  (MySQL, PostgreSQL, SQLite)  **IDEs**:  VSCode (Most popular, doesn’t behave well with Java)  Visual Studio  NetBeans  Eclipse  **Responsive Design:**  In development, dev teams can inspect product through developer tools to test breakpoints and responsiveness for CSS. | **Frontend languages/Frameworks:**  HTML  CSS  JavaScript (React.js, Angular, Vue.js are popular and common frameworks.)  **Backend languages:**  Java  Node.js  **Database:**  SQL databases.  (MySQL, PostgreSQL, SQLite)  **IDEs**:  Visual Studio  VSCode (Most popular, doesn’t behave well with Java)  NetBeans  Eclipse  **Responsive Design:**  In development, dev teams can inspect product through developer tools to test breakpoints and responsiveness for CSS. | **\*Note**  Mobile development is rather niche and requires specific tools.  **Mobile Development Languages:**  HTML  CSS  Xamarin  Swift  Java (fallen out of favor to Xamarin)  JavaScript (React Native)  Objective-C (fallen out of favor to Swift)  **Database:**  SQL databases.  (MySQL, PostgreSQL, SQLite)  **IDEs**:  Android Studio  Xcode  Visual Studio  Eclipse |

## 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**: My official recommendation for the web-based version of Draw It or Lose It is to host the app on Linux servers. Given that Linux servers have a high instance of developer familiarity, low operating costs, open-source contribution to security features, and inter-adaptability with other Linux-based cloud computing options like AWS, I find this is the best solution with releasing Draw It or Lost It to market in a timely and efficient manner.
2. **Operating Systems Architectures**: OS architectures will target popular web browsers. Linux is configured to operate across the board with Google Chrome, Firefox, Edge, and Safari, to name a few.
3. **Storage Management**: As previously mentioned, many options are available with cloud computing, and given that Linux is the operating platform recommendation, storage management should be handled with AWS, where The Gaming Room will be out-of-pocket for storage used on the cloud servers. The Gaming Room can store images on AWS S3 buckets, and using a PostgreSQL database, an open-source and time-tested database - and developer favorite – storage needs should be aptly covered.
4. **Memory Management**: The game images are at the forefront of this concern. The best approach is to compress the images and network configuration to request and get the images, then store compressed images in the user’s browser session-storage cache, to limit CPU load and provide a seamless user experience. Considerations for how each browser stores files in RAM must be accounted for. Chrome is a popular web browser across the market, but is notorious for using the most RAM.
5. **Distributed Systems and Networks**: On the server-side of the application, as previously recommended, a PostgreSQL database instance will handle network requests. Cloud-based Linux servers will effectively manage server in-game instances, and lastly, the client-side browsers will be the consuming portion of the application.

Given current practices, the cloud-native application can be containerized using Docker, an open-source container that will not share the user’s kernel and will spin-up a game instance on the Linux cloud server to serve a game instance for the user.

1. **Security**: Security is a crucial point of conversation for Draw It or Lose It, and can be accomplished effectively using these best practices:

* User Registration and Authentication
  + JSON Web Tokens will be implemented, along with code across the stack to salt and hash passwords for the database. In the case of a database break, user information is much safer as the JWT carries a unique string for the for the password to and from the database.
* Encryption
  + Modern browsers can handle encryption, with some drawbacks:
    - Irregular updates
    - Cookies to store capture user data.
* Given that Linux servers are the choice of operating platform, a high degree of certainty can be expected for security. Regular updates, security experts’ contributions, and long-standing public trust can ensure that user information will be safe
* Regular security audits must be implemented to ensure our server and client code is meeting current best security practices.