

Draw It or Lose It

# **CS 230 Project Software Design**

Version 3.0

## Table of Contents

[**CS 230 Project Software Design**](#_heading=h.gjdgxs)1

[**Table of Contents**](#_heading=h.30j0zll)2

[**Document Revision History**](#_heading=h.1fob9te)2

[**Executive Summary**](#_heading=h.3znysh7)3

[**Design Constraints**](#_heading=h.2et92p0)3

[**System Architecture View**](#_heading=h.tyjcwt)3

[**Domain Model**](#_heading=h.3dy6vkm)3

[**Evaluation**](#_heading=h.4d34og8)3

[**Recommendations**](#_heading=h.2s8eyo1)5

## [Document Revision History](#_heading=h.1fob9te)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/12/22 | Joshua Wozny | 1st draft |
| 2.0 | 11/19/22 | Joshua Wozny | Update evaluation |
| 3.0 | 12/10/22 | Joshua Wozny | Update Recomendations |

## [Executive Summary](#_heading=h.3znysh7)

The Gaming Room produces Draw It or Lose It, a successful Android only gaming application roughly based on the 1980s television game *Win, Lose or Draw*, where teams compete to guess what is being drawn. This project is to develop a web-based version of the game that is playable on multiple platforms. As a web-based application, it will use a client-server architecture. Mac, Windows, Linux, Android, and iPhone environments will need to be available for testing purposes. Each platforms major web-browsers will have to be tested, including Chrome, Edge, Safari, and Firefox.

## [Design Constraints](#_heading=h.2et92p0)

* Browser-based game environment will be utilized. The client-server interactions should use HTTP standard protocols. HTML, CSS, JavaScript will be utilized for web-based client interactions with possible Java based components. Server development may use PHP, JavaScript, or Python dependent on back-end deployment decisions. SQL may be used to interact with database for storing game information on the server, unless a NoSQL database is determined to provide better performance.
* Gameplay performance may be impacted by the web browser being used, as well as the type of device being used. Client side processing should be minimized to prevent performance issues on slower devices.
* One or more teams may play a single game, each team may have multiple players assigned, with names that must be unique and each will be assigned a unique id. This will be accomplished through an Entity base class requiring an id and name for each child class. The Player, Team, and Game classes will each inherit from this base class.
* Only one instance of a game may exist at any given time to maintain consistency of players and games within the distributed environment. This will be accomplished through use of the Singleton design pattern.

## [System Architecture View](#_heading=h.tyjcwt)

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](#_heading=h.3dy6vkm)

To implement the requirements of the Draw It or Lose It applications with the described constraints above, several classes will be required. The classes described herein are detailed in the UML diagram below (Diagram 1). The fundamental classes include ProgramDriver, GameService, Game, Player, and Team.

Of special note, Game, Player, and Team each inherit from Entity, demostrating the OOP concept of polymorphism, used in our application to ensure that every Entity is required to be assigned a unique name and id. Additionally, please note that the GameService class implements the Singleton pattern to ensure that the application meets the design constraints. For development purposes a SingletonTester class is used to test that only one game instance can exist at a time.

* Program Driver includes the main() method, and will be used to drive the application, and contains the method to create an instance of the SingletonTester class.
* SingletonTester is used to test that only one instance of a game exists at any given time.
* Entity is a base class that children classes will inherit from that need a unique name and id. In our application these child classes are Game, Player, and Team,
* GameService manages players, teams, and games. It utilizes the Singleton Pattern such that only one instance of the GameService can exist in memory at any time.
* Each of the following classes provide Encapsulation, an important aspect of OOP design, of each type of object used in the application. Each of these also inherit from the Entity class to guaranty that each has a unique name and id.
  + Game encapsulates all the properties that describe a game and the actions that a game should have available to it.
  + Player encapsulates all the properties that describe a player and the actions that a player should have available to it.
  + Team encapsulates all the properties that describe a team and the actions that a team should have available to it.

**Diagram1**

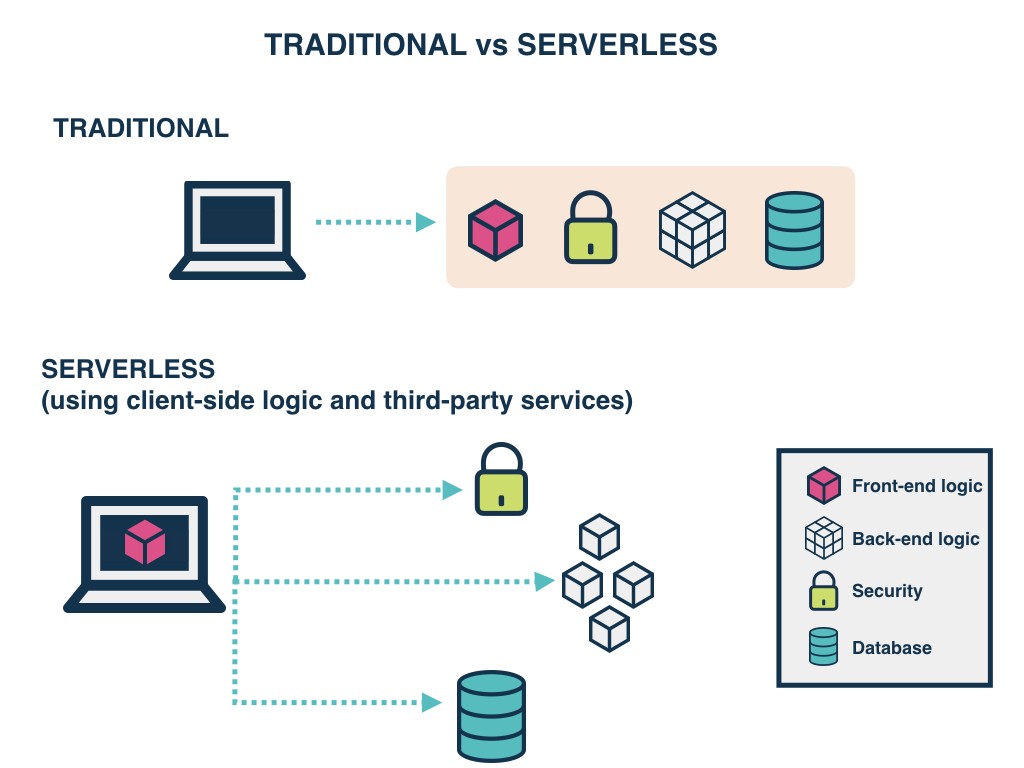
"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' 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](#_heading=h.4d34og8)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Macs provide a powerful server-side solutions with built-in utilities to simplify setup.  **Strengths**: Mac environments are generally more secure, and have built-in utilities to make setup of the server environment easier.  **Weaknesses**: Mac server configurations are typically more difficult to find and are typically more costly. Oftentimes they can also be more difficult to create custom configurations. | Linux provides a powerful  server side solution.  **Strengths**: Linux environments are some of the most commonly available , and lowest cost, server environments and can be customized to meet nearly any server needs. Linux is open source and has many resources available to assist administrators to properly configure. and have built-in utilities to make setup of the server environment easier.  **Weaknesses**: Linux server environments require much greater expertise to operate and configure. | Windows provides powerful server-side solutions that are commonly available.  **Strengths**: Windows environments are generally less secure, and bit like Macs, have built-in utilities to make setup of the server environment easier. There are many Windows users and developers.  **Weaknesses**: Windows server environments are typically less reliable, requiring frequent reboots. Cost is generally higher than other configuration when considering maintenance costs. | Mobile devices can provide limited server-size solution limited to applications that run locally.  **Strengths**: Mobile devices can host their own database for storage of locally deployed application and provide significant security benefits and can provide high performance for appropriate types of applications.  **Weaknesses**: Mobile devices are not designed to be a server resource in a distributed environment. |
|  |  |  |  |  |
| **Client Side** | Mac provides an environment with many development tools that can be used to create browser-based cross-platform applications. Development resources will be moderate as the game already exists as an Android-only application and must be retooled to work as a browser-based game.  Some resources will need to be devoted to client-server interactions that may behave differently over HTTP. | Linux provides an environment with many development tools that can be used to create browser-based cross-platform applications. Development resources will be moderate as the game already exists as an Android-only application and must be retooled to work as a browser-based game.  Some resources will need to be devoted to client-server interactions that may behave differently over HTTP. | Windows provides an environment with many development tools that can be used to create browser-based cross-platform applications. Development resources will be moderate as the game already exists as an Android-only application and must be retooled to work as a browser-based game.  Some resources will need to be devoted to client-server interactions that may behave differently over HTTP. | Developing multiple applications, one for each mobile operating system requires substantial development resources and will require specialized knowledge of each system. |
| **Development Tools** | **Frontend**:  HTML  CSS  JavaScript  possibly Java  **Backend**:  PHP  Python  JavaScript  Java  possibly Objective\_C  possibly Swift  **Database**:  SQL  **IDEs**:  Visual Studio  VS Code | **Frontend:**  HTML  CSS  JavaScript  possibly Java  **Backend**:  PHP  Python  JavaScript  Java  possibly C++  **Database**:  SQL  **IDEs**:  VS Code  Eclipse | **Frontend:**  HTML  CSS  JavaScript  possibly Java  **Backend**:  PHP  Python  JavaScript  Java  possibly C#  **Database**:  SQL  **IDEs**:  Visual Studio  VS Code | **General**:  Development occurs on another platform with the tools provided on that platform (See Mac, Linux, and Windows development tool entries). Each mobile platform has their own specifications and individual libraries. Cross-platform framework and tools are available. |

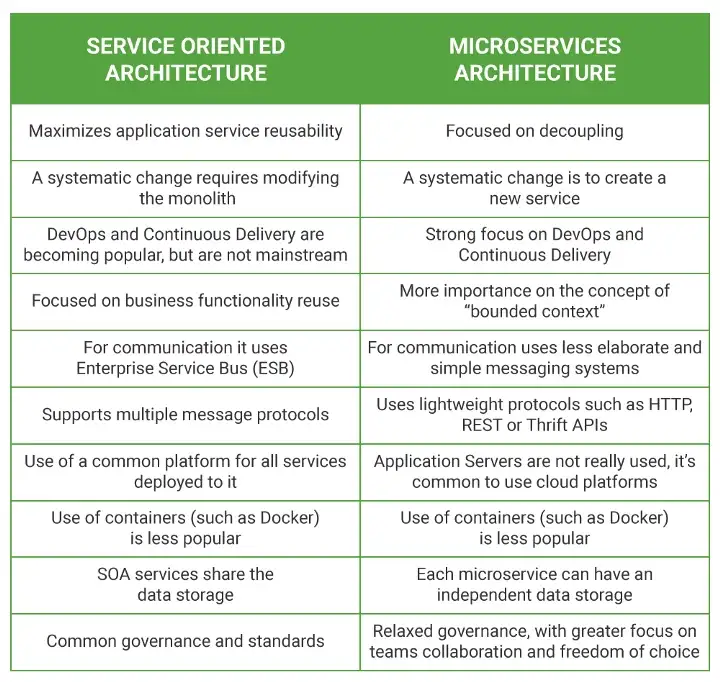
## Recommendations

1. **Operating Platform**: Draw It oe Lose It, an Android-only game produced by The Gaming Room, is being re-developed as a web-based cross platform application. It is recommended that the application be designs with a client-server archictecture using microservices hosted on a Cloud-based solution. Microservices provided include security, file management, storage and transfer, game logic, and game administration.



Source: <https://www.gocd.org/2017/06/26/serverless-architecture-continuous-delivery/>

1. **Operating Systems Architectures**: Each microservice will be cloud-based using the appropriate technology - not dependent on any single operating platform. For instance, a dedicated network file server will be used to maximize read and write speeds by allowing non-contiguous blocks of memory to be used, managing pointers of file locations dynamically. Using a cloud solution for serving files and managing storage allows for additional storage capacity to be added as the application requires it. Similarly, each of the other microservices use appropriate technology suited to the services needs.



Source: <https://medium.com/microtica/microservices-vs-soa-is-there-any-difference-at-all-2a1e3b66e1be>

1. **Storage Management**: In addition to the need for a file server to manage, store, and transfer images. There will also be a need for a database to store and manage other game details for many users, each with one or more player, belonging to one or more teams. This game detail, as well as administrative data, is managed and stored by a SQL database, also housed in the cloud - although some discrete information may be better organized by a NoSQL database instead. By housing the database, or databases, in the cloud the infrastructure can be sized appropriately - with resources being expanded as demand grows. Accessing the data will be managed by the appropriate microservices. To maximize access speeds, cached data may be used.
2. **Memory Management**: Cloud-based micro-services provides for separation of concerns and independent memory management for each service. Using the appropriate cloud-based technology, memory is efficiently managed and should support threading and multi-threading, where appropriate. Memory management can be customized based on how each micro-service will be used.
3. **Distributed Systems and Networks**: Using a cloud-based system mitigates the risks that a system may not be available if there are network or hardware issues - with most cloud solutions automatically adjusting for any localized problems automatically. Maximizing uptime, while minimizing unused resources is one of the major benefits to using cloud services like Google or Azure.
4. **Security**: All components of the application, from the front-end, back-end, and client-server interactions will need to use secure and encrypted protocols. For client-server interaction HTTPS will be required, and 128-bit encryption should be used on all sensitive data. Draw It or Lose It will be designed to use a security micro-service that can be upgraded and maintained separately that will handle authentication and authorization of users. Cloud systems generally provide tools that make using secure protocols easier, and the services provided by cloud systems themselves, are generally secure. However, care must be taken to make sure that security settings are reviewed and adjusted to secure the application, with special care taken to encrypt and secure sensitive personal data.