Draw It or Lose It

CS 230 Project Software Design Document

Version 1.0

Table of Contents

CS 230 Project Software Design Document 1

- Table of Contents 2
- Document Revision History 2
- Executive Summary 3
- Requirements 3
- Design Constraints 3
- System Architecture View 4
- Domain Model 4
- Evaluation 5
- Recommendations 9

Version	Date	Author	Comments
1.0	05/23/25	Justin Crouch	Add summary, requirements, and constraints
1.0	06/07/25	Justin Crouch	Add evaluation for Mac, Linux, Windows, and Mobile platforms
1.0	06/18/25	Justin Crouch	Add recommendations section

The Gaming Room requests that their Android mobile app, *Draw It or Lose It*, be converted to a web-based version. The web-based version should have the same look and feel to the player as the Android version. The web-based version should also have access to the same stock drawings database as the Android version. This version should follow modern application security techniques.

The game contains four rounds. Each round is one minute. Each round displays new stock drawings over the first 30 seconds to help a team of players guess the puzzle. If the team fails at guessing the puzzle by the end of the round, then every other team (in turn) will be given 15 seconds each to guess the puzzle.

Requirements

The following list are the client's **business requirements**:

- Web-based game can be served on multiple platforms
- A game consists of multiple teams
- A team consists of multiple players
- A game has four rounds
- A round lasts 1 minute
- Each round has a puzzle/theme
- Stock drawings related to the round's puzzle/theme are shown to every team
- Stock drawings are additively shown over the round's first 30 seconds
- The current guessing team has the entire round to guess the puzzle/theme
- Every other team is given 15 seconds to make one guess of the puzzle if the current guessing team fails the round
- Players can choose their team name
- Gameplay shows the same stock drawings as the Android version

The following list are the **technical requirements:**

- All game logic is handled by the server
- All players will have a unique name
- All teams will have a unique name
- All running games will have a unique name
- Stock drawings are obtained by the server
- Only one game instance can be loaded in memory
- The Java programming language will be used for the server logic
- The server logic will utilize a REST framework to allow players to interact with the game
- The Javascript programming language will be used for the client logic
- The front-end code should support the most popular browsers
    - Firefox
    - Chrome
    - Safari
    - Microsoft Edge
- The front-end code should render the game depending on the browser's window's size

Design Constraints

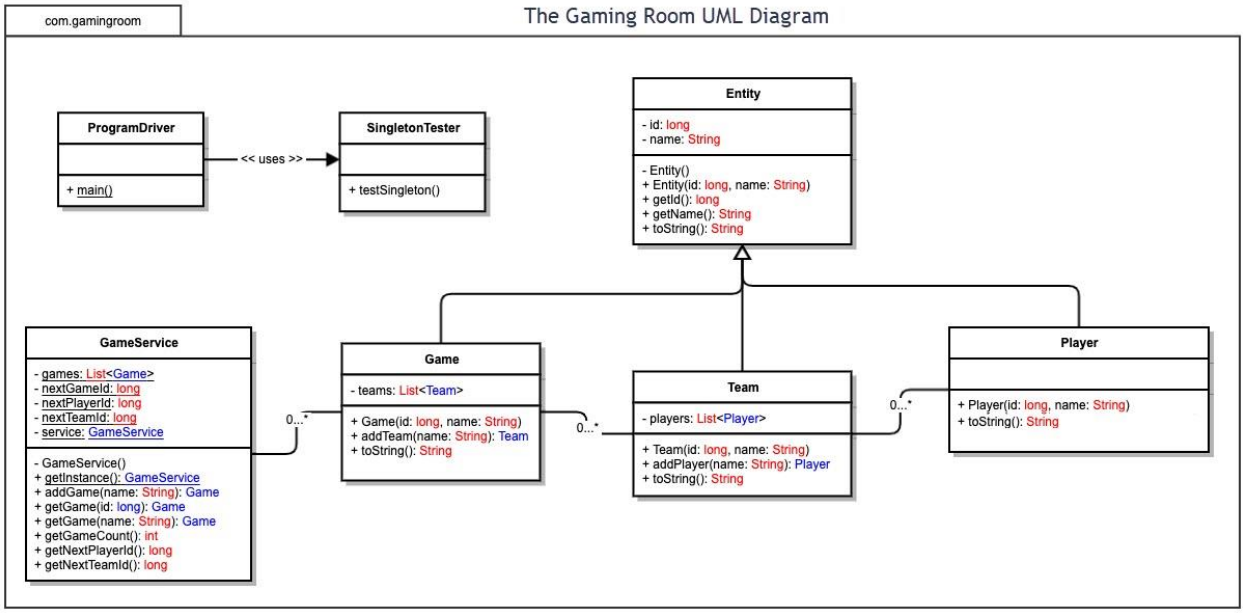
Being web-based, this application requires a separation of “server logic” and “client logic” so that no team or team member can cheat in a round; it also lessens the computation needed on the server, allowing for better server performance and lower server prices.

The front-end (aka “client logic”) will use the Javascript language to render the game to a player in real-time. Other client-based languages can technically be used, however Javascript is supported by the most popular browsers and is easier to build client-side applications with.

The development team must have up-to-date training on the client-server model and the REST framework, including modern security techniques regarding these training points.

System Architecture View

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.



Warning: This portion of the document gets a bit nerdy.

The above image showcases the game’s UML Class Diagram.

The *GameService* class is a Singleton class (i.e. there will only be one *GameService* class in memory for the program’s entire lifetime). This service class has a relation to the *Game* class (i.e. it manages the creation, storage, and access to multiple games on the server). Because of being a Singleton, this *GameService* class fulfills the client’s request of having only one game (management) instance in memory at any given point.

The *Entity* abstract class is inherited from the classes *Player*, *Team*, and *Game*. This relation implies that all entity instances will have an encapsulated id and name set at instantiation, with abstracted methods to get these properties, thus allowing for the client's request of unique game and team names.

The *Game* class stores multiple teams and exposed a method to add a team. This satisfies the client's request of a game having many teams.

Likewise, the *Team* class stores multiple players and exposed a method to add a player. This satisfies the client's request of a team having many players.

The *Player* class is just a concrete entity; nothing special at this point.

Concerns: It is unclear from the current UML class diagram on how teams and players in a game can be accessed. There are no exposed methods on the *Game* class that gets a team by id/name from its list of teams, nor any exposed methods on the *Team* class that gets a player by id/name from its list of players.

Development Requirements	Mac	Linux	Windows	Mobile Devices
Server Side	Source code can only be compiled with a Mac device. MacOS can only legally run on Apple hardware. This means that an Apple product must be purchased to run a server on MacOS, which can be high in initial cost, difficult to scale in the future, and potentially incapable to replace or upgrade hardware (i.e. RAM) on the server. MacOS is considered more secured than other operating systems, but at the cost of lower customization of the OS. Meaning, there may be unnecessary applications and functionalities that cannot be removed from the device or OS that can take up the server's processing power. MacOS has a native backup feature, called Time Machine, that can easily backup data stored on the server.	Linux is an open-source OS framework that does not require specific brand hardware, is generally free to use, and relies on the public community to update. Ubuntu is a popular Linux distribution for hosting servers. The Ubuntu distribution is highly customizable from firewall management to automating scripts. The distribution can be run on very low-end devices. Because Linux is highly customizable, it requires expertise to properly secure the OS.	Windows is a licensed OS, meaning the OS must be purchased to use it. Due to its popularity, Windows devices are frequently attacked with malware. The OS, being proprietary software, typically contains bloat-ware from its vendor that cannot be disabled or uninstalled. Many powerful development tools are supported on Windows. Although Windows devices are frequently attacked, the OS is regularly updated.	While technically possible to host a server, mobile devices generally have low-end hardware. This limits how many requests the server can process.

Client Side	Since the application should be deployed via a web browser on desktop clients, the application should be tested on the Safari web browser. This is the default web browser for Mac users. Since the Safari web browser is no longer supported on non-Apple products, a Mac device will need to be purchased for development, which can be costly. Ensuring the application is displayed correctly on different screen resolutions should be considered	The application should be tested on the Firefox browser since this is the default browser for many Linux distributions. Firefox is free and is supported on most operating systems. Again, consideration of different screen resolutions should be made.	The application should be tested on the Microsoft Edge web browser since this is the default browser for Windows products. Microsoft Edge is free and supported on Windows, Mac, iOS, and Android devices.	Expertise in Android and iOS app development will be needed. Even though the client already has the Android application built, it will most likely need to be altered so to properly interact with the server. Developers will need to know about modern security techniques regarding iOS and Android platforms. Also, a yearly subscription on the iOS app store and one-time fee on the Android app store will be required to publish their app on the respective stores. A Mac device is required to develop iOS apps.

Development Tools	For the server side, the Java programming language can be used without the need of compiling server code on a Mac device. However, to run java applications on Mac, the Xcode development tools must be used. Maven is a good tool for managing library dependencies for Java applications. For the client side, JavaScript would be most appropriate to request data from the server and to build the application logic with. HTML would display the game to the user, and CSS would add styling to the display. A specific IDE is not required.	The Java programming language can also be used on Linux for the server since the Java Runtime Environment is supported on Linux. Since most modern web browsers support JavaScript, JavaScript can be used for the client logic of requesting data from the server and updating the client's display. HTML and CSS will be used for the client's display. A Javascript framework, like Vue.js, can be used for building the client-side with maintainability in mind; in this case, the Node.js runtime tool would be required for development.	The Java programming language can certainly be used for the server. Maven can be used for managing library dependencies and building the server. Git should be used for version control of the source code. JavaScript, HTML, and CSS will be used on the client side. A JavaScript framework, like Vue.js, and a JavaScript runtime environment, like Node.js, can be used to build scalable, maintainable, and functional client web views.	For iOS, the Swift programming language and SwiftUI framework would be used for the client-side logic and interface, respectively. The Xcode development tools must be used to compile apps for the iOS app store. For Android, the Android Development Kit must be used to compile apps for the Android app store. The official IDE for Android development is Android Studio. Java can be used for the client application. The current Android application should be analyzed for interacting with the server.

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**: The Ubuntu OS running the Draw It or Lose It server as a Java application is recommended.

2. **Operating Systems Architectures**: Ubuntu is a free and open-source Linux distribution which has popularity in server environments. The OS can be easily deployed on an on-premises server or on a cloud environment. Multiple user accounts with customizable and manageable permissions can be created within the OS, and multiple users can log in remotely to the OS for maintenance or development.

   Java is a popular interpreted programming language that allows source code to be "compiled" once and run anywhere that supports the Java Virtual Machine. To manage library dependencies and ensure consistent builds across development environments, the Maven build tool should be used. The server should be developed using the Spring Framework, which provides a solid backbone for creating a web application using Java.

3. **Storage Management**: To store user data, a PostgreSQL database should be used and stored on a Solid-State Drive for fast, structured access to the data. Images should be stored on an SSD as well for the same reason.

4. **Memory Management**: By only allowing one game manager instance per server execution, the separate game instances running on the server can be managed using Object Pooling. This technique would pre-allocate memory space for game instances to utilize at the start of the program execution, thus preventing unnecessary and time-consuming garbage collection of allocating and deallocating memory when game instances are started and terminated. The same technique can be used in handling multiple teams and players.

5. **Distributed Systems and Networks**: The server should be built using a RESTful architecture. By using this architecture, only data corresponding to a network request is sent to the connected client; it is the client's job for parsing and rendering the data. This allows for a separation of data and logic/UI, providing an easier path to communicate with the server from any client.

   To allow for faster downloading of the game's images, a Content Delivery Network (CDN) should be utilized for serving only the images. Similarly, the user database should be stored on a separate server and accessible only by the Draw It or Lose It application.

6. **Security**: The Draw It or Lose It application should be executed with a user account that is given the least amount of privileges needed for proper execution; if a vulnerability exists in the application that allows for remote code execution or a directory traversal attack, this would drastically lessen the impact of these attacks. Any user input requested by the server and client, including request headers and form submissions, should be white-list validated, properly character encoded, and verified for containing all requested fields. Any user input supplied to the user database must be parameterized to prevent SQL injection. All endpoints of the server

must only allow authenticated requests and be limited to the least amount of privileges needed for functionality.