Draw It or Lose It

CS 230 Project Software Design Template

Version 1.2

Table of Contents

CS 230 Project Software Design Template 1

Table of Contents 2

Document Revision History 2

Executive Summary 3

Requirements 3

Design Constraints 3

Domain Model 4

Evaluation 5

Document Revision History

Version	Date	Author	Comments
1.0	01/24/21	Brad Jackson	Initial proposal
1.1	02/07/21	Brad Jackson	Added Evaluation
1.2	02/21/21	Brad Jackson	Added Requirements

Executive Summary

Draw It or Lose It hopes to recreate the classic gameplay of Draw It or Lose It (currently available on the Android platform) in a web browser. One team will attempt to guess an image which is effectively drawn by the application. After thirty seconds the drawing will be complete and other teams will have fifteen seconds to guess what the image is.

Requirements

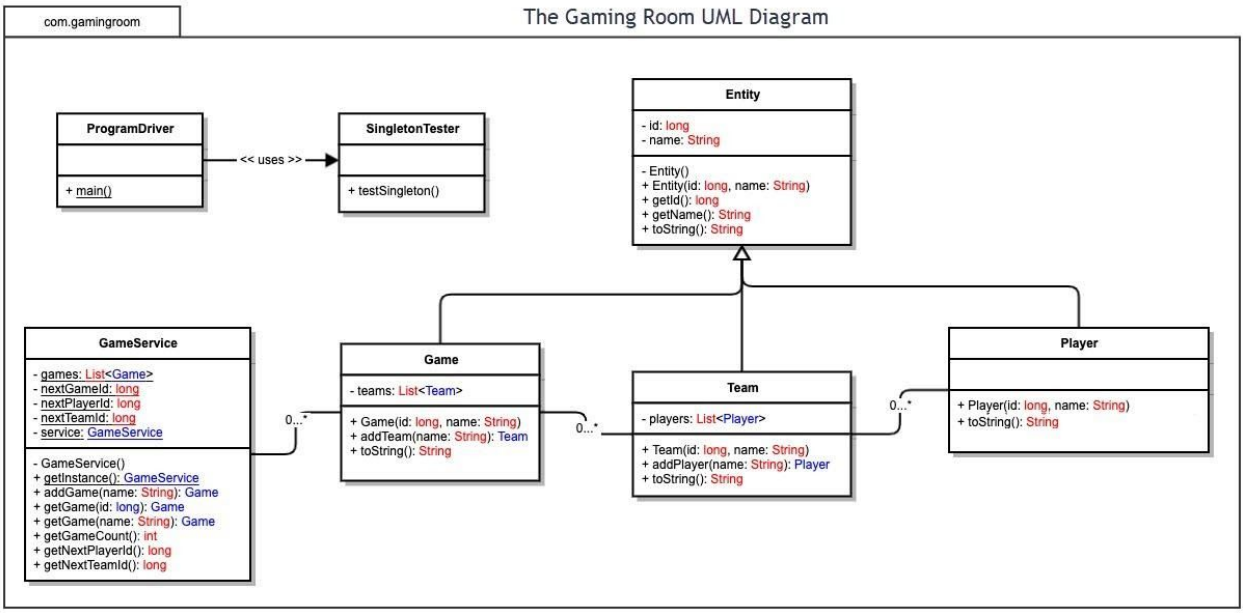
For the operating platform of the Draw It or Lose It web service we will be using an Alpine Linux. The reason for this is the reduced clutter and easy-to-configure security of this particular flavor of Linux. Ideally we will host this operating system within a Docker container housed within a Kubernetes platform as a service (PaaS). Because of our data size, binary large object (BLOB) storage could be leveraged until we have more images. The container would interface with the BLOB storage by virtually mounting it and using a memory-mapped architecture via the Network File System (NFS) protocol. Images will be hosted outside of the host operating system permanently and only renders will be done in real-time. For security we should also consider encrypting the data at rest. In terms of security, if we are using Microsoft's Azure platform then we can rely on role-based access controls and OAuth 2.0 to delegate access to the web service endpoint. By enabling scaling we can ensure the availability of our service is 99%. When the containers scale they are able to share compute load and network requests are managed by a load-balanced ingress controller. We can provide a secure, scalable, and cheap web service using cloud architecture and minimizing our compute and storage footprints.

Design Constraints

The application will have to have a fairly low latency between the servers and clients because the game is based on real-time rendering. A game will have the ability to have one or more teams involved. Each team will have multiple players assigned to it. Game and team names must be unique to allow users to check whether a name is in use when choosing a team name. Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

Domain Model

The ProgramDriver class contains the main method of the application and uses the SingletonTester class to test that there is only one instance of the GameService class. For each GameService (if any) there may be multiple Game classes. A zero to many relationship is shown below between the GameService and Game classes. There may also be zero or more Team classes within a Game and zero or more players in a Team. The Game, Team, and Player classes all inherit from the Entity class. Along with the inherent variables and methods of Entity these classes also have variables and methods specific to their namesake.



Evaluation

Development Requirements	Mac/OSX	Linux	Windows	Mobile Devices
Server Side	The cost of hosting our web server on OSX would be significantly higher than Linux or Windows; however, the principle would be the same (configuring tomcat or another web server).	Web server hosting will require minimal configuration depending on our deployment method and would be cheapest.	Windows offers in-house web server hosting via IIS; however, the license will cost more than Linux which is free to use.	It would not be reasonable to host a web server on a mobile device. I am sure we could figure out how to host a k8s cluster on Android; however, it would be time consuming and iOS certainly will not allow such freedom.
Client Side	Safari is the native browser within OSX so we will need to make sure our web application is compatible with it as well as the other commonly-used browsers.	Firefox and Chrome will be the major browsers used within the Linux environment.	Microsoft Edge is the native browser of Windows so we will have to have compatibility with it as well as other commonly-used browsers.	Chrome and Safari are the two biggest contenders for mobile browser users. We should focus our efforts on optimizing for these on mobile platforms.
Development Tools	We will most likely have to use XCode and swift if deploying natively on OSX. We should avoid the time sink if possible, choose an IDE (VSCode or Eclipse), using Java, and deploy on Tomcat. There would be no license costs.	Assuming we are using a GUI-enabled Linux development environment we should employ VSCode and Java, deploying Tomcat. Deployment targeting a Linux container would be the most effective in terms of scalability and cost. We can take advantage of PaaS such as Amazon or Azure's Kubernetes (k8s) cluster management. There would be no license costs.	We can use VScode, Java, and Tomcat for our development pipeline; however, it may be useful to research the advantage of using IIS and C# to determine if there are cost savings involved; however, there are license costs involved in using IIS so it would probably not be worth it.	We would need to use XCode and Swift on another operating system (OSX, *nix, Windows) for iOS or Android Studio and Kotlin/Java if we were going to host a webserver from a phone. It would not be worth the cost despite having no license costs.