Draw It or Lose It

CS 230 Project Software Design Template

Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	03/19/23	Benjamin Huffman	Completed initial software design template for Draw It or Lose It and provided recommendations for The Gaming Room
1.1	04/02/23	Benjamin Huffman	Completed development requirement table

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

The Gaming Room has requested a web-based version of their Draw it or Lose It game. The game is currently only available on Android and involves teams guessing a rendered drawing before a 30 second time limit expires. The game should allow for one or more teams to play at a time with multiple players on each team. Game, team, and player names and identifiers must be unique to avoid conflicts. Only one instance of the game can exist in memory at any given time.

CTS proposes developing a web-based application using a Java backend, a javascript framework front-end (React, Vue, Angular, etc.), and a document based database (MongoDB) for storing the stock images as well as scores, names, and game results. We will use unique IDs and names for each game, team, and player instance. We will iterate through existing games, teams, and players to ensure names are unique and use an incrementing ID approach to ensure no two IDs are the same. This will allow multiple teams of multiple players to participate.

This solution ensures The Gaming Room will receive a scalable application that will work cross platform and meets all success criteria.

To continue with the process we request the collection of stock images used in the existing game to incorporate into the web-based version as well as any additional design constraints not outlined in this document.

Requirements

Original Functionality from the existing Android application must be preserved
Must be a web-based game
Each game, team, and player instance must be unique
Must be able to have multiple teams of multiple players
Only one instance of GameService may exist at a given time

Design Constraints

Must be web-based
Cross-browser compatibility must be ensured
Must be secure
All original functionality must be carried over from existing Android application
GameService must follow singleton pattern
Iterator pattern must be followed for adding new games, teams, and players
Application must be scalable handle an increasing number of teams and players

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.

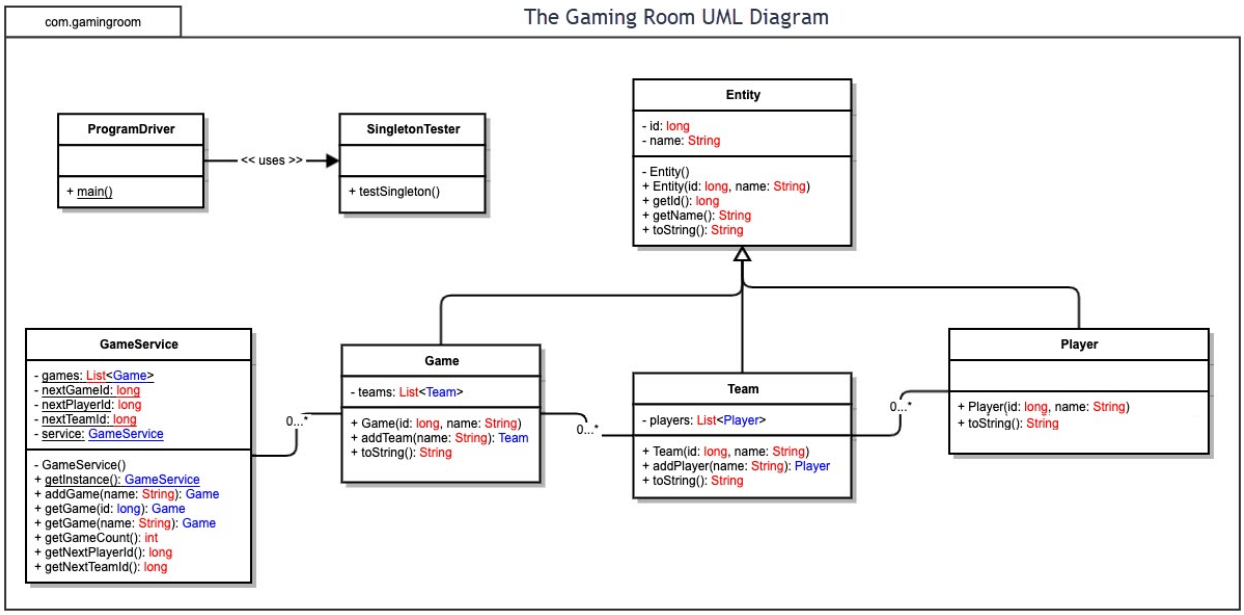
Domain Model

The UML diagram provided below shows the gamingroom package.

The inheritance from Entity to the Game, Team, and Player classes is shown. The Entity class will encapsulate the id and name of its subclasses and provide access with the accessor functions getId() and getName(). Game, Team, and Player will extend the Entity class and inherit the id and name of the superclass and as well as the accessors for these fields. Each subclass will have an override for the toString() method for a clearer representation of which object is being cast to string.

The GameService will encapsulate the next ids for use in the addGame(), addTeam(), addPlayer() functions and will be associated 0 to many to the Game class as it will be able to instantiate many instances of the Game class. This same association is carried from the Game class to the Team class through the addTeam() method and from the Team class to the Player class through the addPlayer() method. The Game and Team classes will only be able to access the appropriate nextId field of GameService through calling the getInstance() GameService method and calling the accessor for the appropriate field due to GameService encapsulating this field.

The program driver will house the main method and drive the application and uses the singletonTester app to ensure singleton functionality of the singleton classes in this package.



Evaluation

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	Mac is built on Unix so it is usually reliable for hosting web-based applications and has a wide range of development tools in its arsenal. Can be expensive hardware wise but only one machine is 'required' for hosting.	Linux is reliable for hosting web-based apps and has a wide range of development tools. It also has the benefit of being free and open source. Linux can be modified to meet specific needs. It can be difficult to use without experience and know how however.	Windows is the most popular desktop os. It is traditionally more user friendly and can easily be integrated with other Microsoft products like azure. Very user friendly considering the OS's popularity and presence in the market.	While common and portable they also come with processing power limitations as well as battery constraints. May not be the best option for web hosting.
Client Side	Mac hardware is generally more expensive than other options. Software must be tested on Mac devices to ensure functionality which can drive up price and time considerations. Mac applications can be listed on an existing website instead of directly through the Mac app store so no additional hosting costs are required for a Mac client app.	Lower cost due to free and open source OS. May be time consuming to test distributions on multiple platforms and hardware. Requires knowledge of linux systems and features. The client-side application can be hosted on an existing website for download on linux systems for no additional cost (other than cost of operation of the webpage.)	Several different versions of Windows may need to be supported driving up time and cost. Lowest desktop expertise barrier client side as it is the most popular desktop OS. Can listed for download on an existing website so no additional hosting cost will be added for offering this client side app.	Can be developed on other operating systems which lowers cost but development is usually different than native development on each platform. Must be familiar with many different hardware and software limitations as mobile comes in many flavors. Hosting the application on the Apple App Store does cost a licensing fee of $99 a year. It is assumed the existing android application is already hosted.
Development Tools	Java, React, NodeJS, Eclipse and VSCode can solve all problems for this type of software since it is web-based. Docker can be used for containers. Container images for specific environments can be used to develop for other operating platforms. Additional client-side Mac native development tools like Xcode and home-brew can be used for native client application development.	Java, React, Eclipse, NodeJS, and VSCode can solve all problems for this type of software since it is web-based. Docker can be used for containers. Images can be used to develop clients for specific environments if all client side development takes place in linux.	Java, React, Eclipse, NodeJS, and VSCode can solve all problems for this type of software since it is web-based. Again Docker can be used for container management and assist in developing for other platforms.	Java, React, Eclipse, NodeJS, and VSCode can solve all problems for this type of software since it is web-based. Swift can be used to develop iOS native applications. Android studio is a great tool for developing android specific applications but this has already been taken care of by Game Room.

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**: I personally would recommend a remote web-based platform for deployment of Draw it or lose it. A nodeJS/express based server deployed by Heroku, AWS, etc. with a mongoDB database will allow for scalability, cross platform compatibility, and ease of use.

2. **Operating Systems Architectures**: This style of approach would be considered monolithic architecture. The server-side code, database, and client application are all packaged together and deployed by the same hosting service.

3. **Storage Management**: One potential storage management option is MongoDB. This is a document based database and would allow for scalable and organized data management. SQL and other structured databases could be used but I would recommend the more flexible document based approach provided by MongoDB.

4. **Memory Management**: Thanks to the Java backend a lot of the memory management will be done implicitly by the JRE garbage collector. This will ensure that all unused game, player, and team instances are cleaned up and removed from memory when no longer in use.

5. **Distributed Systems and Networks**: Hosting the application server with a cloud/web-based solution allows for one connection point (a url provided by the Gaming Room) that any device with a web browser can access. With many servers nationwide reputable hosting services like AWS will likely have processes split between many servers and should an outage occur in one area a loss of connection or an outage will be mitigated by switching processes to another server.

6. **Security**: AWS and other web hosting uses secure connections and allows for SSL/TLS certificates to ensure security. We can take additional measures to encrypt usernames and passwords for authentication and to store only hashed values in the database. Web hosting also allows for us to deploy updated versions of software in a continuous fashion to update dependencies to the most recent and most secure version as they are released.