< 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	<mm/dd/yy>	<Your-Name>	<Brief description of changes in this revision>

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

Gaming Room wants to develop a web-based game “Draw It or Lose It “, that serves multiple platforms based on their current game. The application will render images from a large library of stock drawings as clues, and teams will guess in one minute. If the team does not guess the puzzle before time expires, the remaining teams have an opportunity to offer one guess each to solve the puzzle with a 15-second time limit.

Requirements

- 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.

Design Constraints

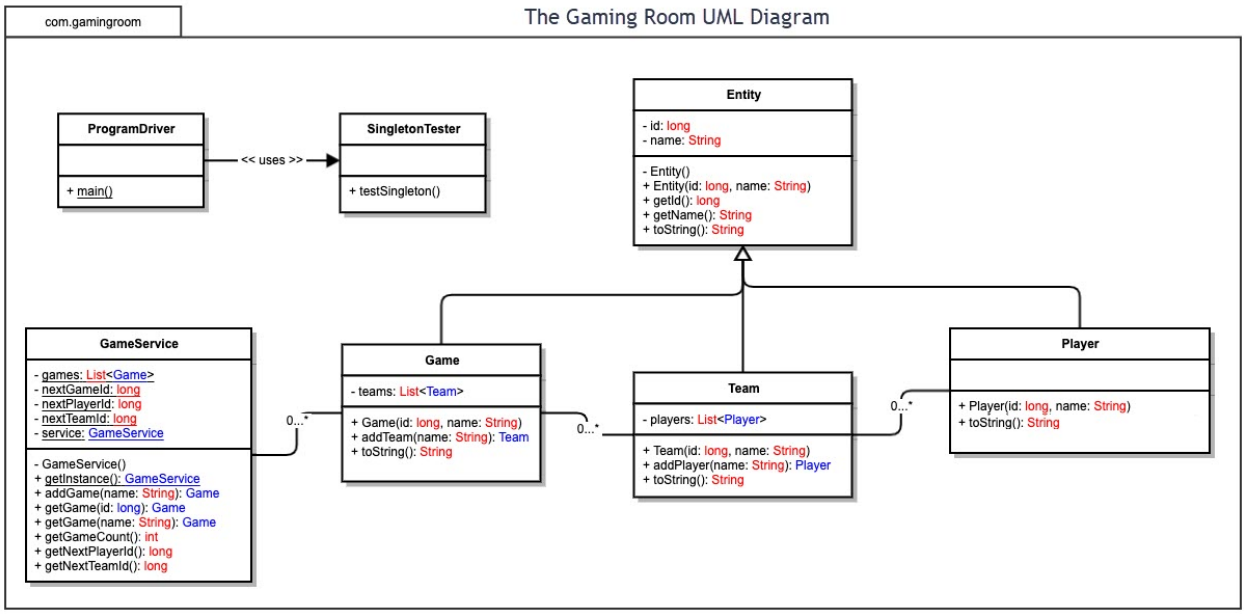
- Need to save all the image in Databases that will used as clues.
- Game should be able to fetch random images from database and use them as clues.
- Game should support multiple teams and each team can have multiple players.
- Games and teams should have unique names so that they can searched uniquely using name.
- All the meta data of game (Teams, Players, points scored, which is active team to response, etc) should be saved in databases for persistence. This is required to restart the game in case of restart.
- 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.

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




```
UML Explanation

Entity is base class.
Entity base class is inherited by Game, Team and Player class.
GameService is a Singleton class and provides single instance of GameService.
Its also responsible for generating unique for Games, Teams and Players.
GameService has 0 to many association with Game class. GameService manages
the lifecycle of Game objects and is responsible to create and delete the
Game instances.
Game class has 0 to may association with Team class. Game manages the
lifecycle of Team objects and is responsible to create and delete the Team
instances.
Team has 0 to may association with Players. Team manages the lifecycle of
Player objects and is responsible to create and delete the Player instances.

Class blueprints that is created out of UML diagram can be found in the
source code.

Source code is completely inspired from UML diagram with few exceptions:
I have added a constructor to Game, Team, Player classes that only take
String name as argument. The constructor will call respective getNextId
functions from GameService class.
This has been done so that class is only one responsible to generate UIDs and
not the class that is creating the instance.
```

Evaluation

Development Requirements	Mac	Linux	Windows	Mobile Devices

Server Side	**Cost:** It is most expensive solution for server deployment. It has most expensive hardware as user can only use Mac machines and one of the most expensive licenses for all the software that will be required for SDLC and maintenance. Developmental cost is also expensive as Mac developers are expensive compared to Windows. **Stability/Performance**: Mac based servers are based upon BSD/Unix and are considered stable, high performance and well suited for long running systems. **Ease of use:** Mac supports both GUI and terminal based interface for users and programmers which makes it very easy to use for server development, deployment and maintenance.	**Cost**: It is least expensive solution for server development. There are no licensing fees for any software. The only possible cost is servicing cost if REDHAT is the chosen OS but is not mandatory. Development cost is moderately expensive as it needs advanced developers but there is good availability of developers in market. **Stability/Performance**: Linux based systems are battle tested systems and are considered as holy grail in terms of performance and stability. These systems are designed to scale horizontally and vertically further increasing the performance. **Ease of use:** Linux is considered most unfriendly OS as terminal is preferred choice to interact with the system. Though it has GUI but is not considered mature when compared to MAC or Windows.	**Cost:** It is less expensive than Mac and more expensive than Linux because of licensing fees of all software. User is free to choose any hardware as windows support most of hardware. Development cost is least as compared to other systems. **Stability/Performance**: Windows is least trusted servers because of historical reasons but recently they are gaining good feedback in terms of stability. But still it will last choice in terms of stability. **Ease of use:** It's the best when it comes to usability as everything is GUI based and that makes development/deployment maintenance very easy.	**Cost:** No one has ever used mobile for server deployment as mobiles are not designed to be server machines. But still someone wants to try, they needs to make array of mobile devices and write own software from scratch to create system that can support server development on mobile devices. So mobile device is not applicable for server development/deployment.

Client Side	Most expensive choice for development as development is done on Mac machine and License cost development Id is quite high. Mac developers are quite expensive as they are less available in market. Time to develop is relatively ok as there is lot of developer support from MAC, it has its own cost.	Least expensive development cost because there are practically no licensing fees. It needs advanced developers, but the developers are easily available in market. But Linux is least preferred platform for game client development because of least penetration among the individual customer base.	Relatively expensive because of licensing cost of IDEs and other software required for client-side development. Time of development is relatively low because of enormous amount of development support from Microsoft as well as on the internet. It is one of the most desired platforms as it has lot of individual customer base.	Cost depends upon the choice of OS. If it IOS, then it has similar issues as MAC in terms of cost, time, and ease of development. If the OS is android, then it's almost free as to develop and vast developer support from Google and other vendors. The only cost that needs to be considered is the fees for marketplace.
Development Tools	Swift or Objective C programming language and Xcode IDE are preferred choice. But it also supports C, C++, Java, Python, etc as MAC supports the compiler tool chain of all the popular high-level languages	Its practically support all the high level compiled of interpreted languages. Its user choice to develop in C, C++, Java, Python, etc and practically everything is supported.	Supports all the popular languages and its VS code IDE is amazing. But preferred language for development is C++ and C#.	Swift is used for IOS development and Xocde ide provides the complete tool chain for cross platform development/testing. Java/Eclipse or IntelliJ are preferred language/IDE for app development.

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.

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**:
    - After Evaluating MAC OS, Windows OS, Linux OS, and Mobile OS, I recommend **Linux OS** for game "Draw it or Lose it"

2. **Operating Systems Architectures**:

    - Linux servers are least expensive choice. There is no cost for Operating servers and all the softwares needed for development and maintenance are completely free.
    - Linux servers has lightweight kernel, and it allows advanced user to configure the OS depending upon the requirement. This allows user to include minimal drivers and packages. This advanced tweaking allows load time and runtime of linux kernel to be extremely fast and highly stable.
    - Because of Opensource, all the major companies are actively developing and testing linux OS. That helps it to be always latest and bug free. This also explains that why Linus servers are so stable.
    - Linux servers are most stable and considered fastest operating systems.
    - It has least amount of downtime as compared to any other OS and helps better user experience.
    - Linux servers are hot upgradeable I.e they don't need restarts in case of upgrades.
    - Linux servers can be scaled vertically with ease by adding more RAM and storage as the need arises.
    - Linus servers has excellent support for Virtual machines and docker containers.
    - Linux OS is considered the most secure OS. The reason is because of it being open source. Since it has open source, security researchers all over the world constantly tries to find flaws in OS and constantly fixing it.
    - All the commonly used programming languages are fully supported along with the complete support for IDE and SDK tool chain.

3. **Storage Management**:

    Linux has excellent support for various Filesystems and ext4 is most considered most reliable file storage management. So we will use ext4 filesystem to save all file on system.

    Apart from this, we will use MYSQL database system to store images.

For better network performance, each image will be divided in 30 chunks. This will help server to send minimal data to client and it will enable server to support multiple concurrent sessions.
Images (chunks) will be stored in mysql table and the primary key of the table storing images will be image_name+chunk_number. It is extremely important to have properly selected primary key as it creates indexes in Database for faster and efficient search.
We will run mysql in master slave configuration so that we can have data replication and high availability.

We will also use a caching mechanism (Memcache or Redis) so that any request to fetch an image chunk should be first tried at cache server, and if not found in cache server, then fetch it from DB server, saved the fetched image chunk in cache for future access, and then send the result back to client.
This will enable faster access to data as mysql request are much slower as compared to data being fetched directly from DB server. Caches are faster because they save all the data in RAM.

4. **Memory Management**:
   At server side, it has 3 independent components that uses memory:
   1) Server instance responsible for making TCP connections with clients
   2) Caching server instance will be responsible for caching images chunks, once fetched from DB.
   3) Mysql DB server that is responsible for saving images persistently on file system.

   Server instance memory requirement will be max number of TCP connections.

   We would like to provide maximum possible RAM to caching server. In ideal condition, we would like to have all images chunks in cache server. So, cache server should be allocated 200 images * 8mb * some extra ram of internal data structures.
   Almost all the Caching system use Hash tables and LRU cache implementation. Hash tables are needed for faster key value lookup and LRU cache (doubly linked list) for keeping most recently used at head and least recently used towards the tail of linked list.

   The RAM required by server instance will be MAX_NUM_TCP_CONNECTION * memory needed per tcp connection * one image chunk size (200/30mbs) + some extra RAM.

   MySQL DB server will need to save table index in RAM a d approach's size of index table will be BTREE_NODE*num_rows_in_table(200) + some extra memory for mysql server.

5. **Distributed Systems and Networks**:
   We add the complexity of distributiveness in system to achieve horizontal scalability and High Availability.
   Horizontal Scalability means that number of instances serving the client requests can grow as the number of incoming requests grows.
   High availability means that service never goes down or it has very minimal downtime.

   Steps to make game application distributive.
   Client application talks to server using Rest API interface using HTTP protocol. Server treats each request stateless.
   Server talks to caching service using caching protocol on TCP connection.
   Server talks to Mysql DB service using TCP(JDBC) connection and use SQL query to talk to mysql server.

   The Server instance behind the Api gateway so that game clients only make connections with API gateway.

   This makes client service loosely coupled to server service. Then client only needs to have public api of the api gateway and that is it.
   Server service can independently scale up and down behind the Api gateway. Api gateway also does the load balancing of requests to server service. Api gateway also act as SSL termination and make simple HTTP connection with service which is behind the protected network. API gateway also does rate limiting in case of DDOS attacks.

   Now server needs to talk to Cache service. We will again put cache service behind the api gateway so that we can get advantage of dynamic scaling of application, SSL termination, rate limiting, etc.
   In simple implementation of cache service, we can calculate the hash(request)/num_cache_server and that will decide which server to store/retrieve the results. But this cache becomes inconsistent in case when one of the cache servers goes down or a new cache server is added.
   So, recommend design is to use Consistent hashing approaches because when one server goes down, it only impacts the cache entries on the adjoining servers of the failed server and all other servers behave normal. And new servers can be added without affecting any existing cache entries.

   One server instance can't find result from cache service, then server needs to fetch result from database server.
   So again, we will put DB service behind the api gateway.
   Then we will use sharding of database depending upon the hash(imagename_chunkid). After sharing, we will have master/slave configuration for each shard. It will guarantee high availability of mysql server as if master is down, slave can process the request.

So from above, we saw different techniques that are used to make any small service distributive in nature.

6. **Security**:
   SSL/TLS is the backbone of encrypted communication between any two services. So, connection between game client and server instance, server instance and cache instance and server instance and db instance has to be HTTPS connection.

   Database that saves username and password has to make sure that all the passwords should be saved in database in the form of SHA256(Userpassword + cryptographic salt). That will prevent user password leakage in case of db server getting compromised.

   Since every client connection needs to be validated, but game client makes several requests in one game session, and server should not ask for username/password on every request. So, server instance should generate server tokens (JWT tokens) with appropriate TTL so that server can validate request just by looking at tokens.

   Apart from this, Linux servers have hard disk encryption. Distributive designing above give security from DDOS attacks. Linux also supports Firewall support.

   REFERENCES:
https://www.youtube.com/watch?v=6ULyxuHKxg8&t=5s&ab_channel=ByteByteGo
https://wikitech.wikimedia.org/wiki/API_Gateway
https://en.wikipedia.org/wiki/Redis
https://en.wikipedia.org/wiki/Memcached
https://www.mysql.com/why-mysql/
https://www.youtube.com/watch?v=j9QmMEWmcfo&t=40s&ab_channel=ByteByteGo
https://stackoverflow.com/questions/536584/non-random-salt-for-password-hashes/536756#536756
https://docs.kernel.org/