Draw It or Lose It

CS 230 Project Software Design Template

Version 3.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	09/20/24	Aidan Farhi	Initial draft.
2.0	10/06/24	Aidan Farhi	Updated the Evaluation section.
3.0	10/20/24	Aidan Farhi	Updated the Recommendation section.

Executive Summary

The Gaming Room, a gaming company, is seeking to build a web-based version of their Android game, Draw It or Lose It. The game must be ported over to an application that can be accessed through a web browser at a publicly available URL. It must have the same functionality and look-and-feel as the mobile version.

Requirements

Business Requirements:

- The web-based game must have the same functionality as the mobile version.
- The web-based game must have the same look-and-feel as the mobile version.
- The game must support multiple games, teams, and players on each team.

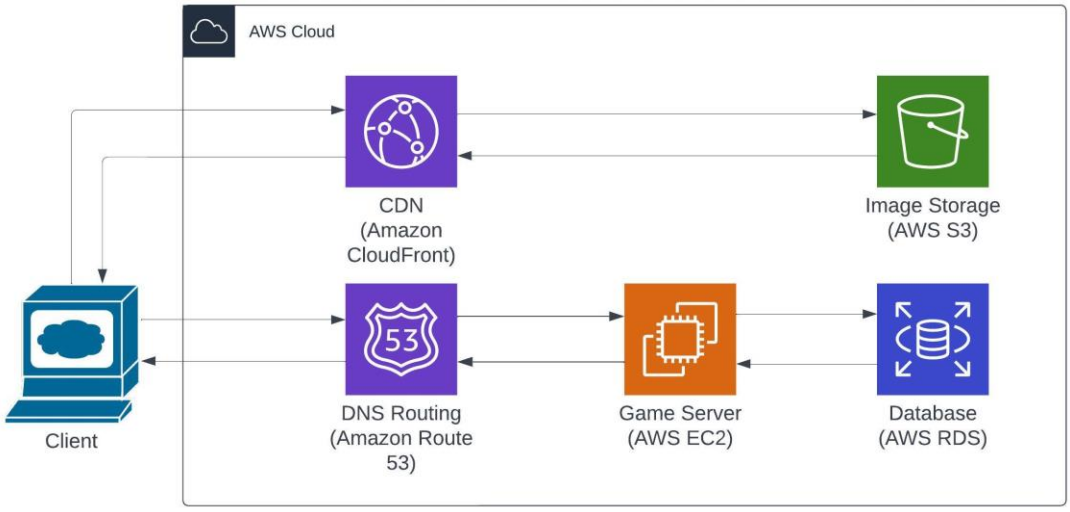
Technical Requirements:

- The application needs to be deployed on a server that can receive and respond to requests over the internet.
- The application needs a database to store user, team, player, and game information.
- The application needs to store a large library of images.
- The application needs to serve images.

Design Constraints

This solution will require highly skilled engineers that are able to translate the existing game logic, user interface, and user experience from a mobile application to a web application. Since this solution needs to be available via the web, a cloud-based hosting solution, like AWS, should be leveraged for scalability, flexibility, and speed of development. A language that is portable, stable, secure, and offers a range of frameworks for application development should be used. This will enable the engineers to quickly iterate, scale, and deploy the application on an arbitrary range of platforms. The engineers that are building the application should be proficient in mobile, front-end, back-end, and cloud engineering. If not, engineers specialized in each area must be brought on the team. This could present an increased cost in terms of headcount. An efficient way to store and serve the large library of images needs to be implemented. The images will present a significant amount of data, which can increase storage costs. A content delivery network should be used for fast and efficient serving of images to clients globally.

System Architecture View

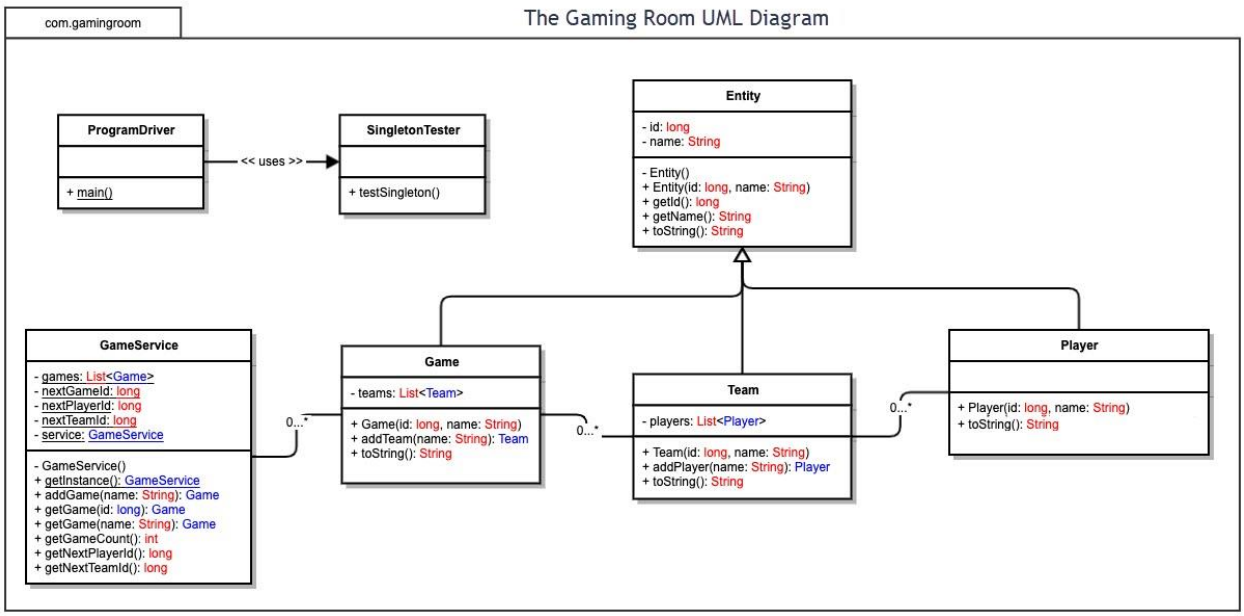


This diagram shows an example of an AWS cloud-based system architecture. Amazon CloudFront is used for content delivery of images and AWS S3 is used to store the images. AWS EC2 is used as the Virtual Machine on which the web application will run. AWS RDS is used as a database and Amazon Route 53 is used for routing requests.

Domain Model

This diagram exhibits the classic OOP design principles of inheritance, polymorphism, abstraction, and encapsulation. Inheritance, which encourages code reuse, is shown by the Entity class. The Game, Team, and Player class all extend the Entity class. All common attributes and methods are defined in the parent class and are available to each child class. There is encapsulation used by the GameService, Entity, Game, and Team classes. They all have private members that are only accessible through public methods, hiding their internal state while providing controlled access to their functionality. The GameSevice class also uses polymorphism by implementing method overloading. It implements two different versions of the getGame() method, each one accepting different parameter types. The Entity class exposes a simplified interface for interacting with different entities within the system, an implementation of abstraction.

All the relationships between the objects in the system are highlighted, giving an overview of the system's functionality. The ProgramDriver instantiates and uses the SingletonTester class. The cardinality between the GameService, Game, Team, and Player classes is notated, showing that there are 0-to-many relationships between them. Inheritance is notated by the arrows flowing from the Game, Team, and Player classes to the Entity class. Overall, this UML design diagram presents an understandable and efficient way to model and implement the software system.



Evaluation

Development Requirements	Mac	Linux	Windows	Mobile Devices
Server Side	**Characteristics:** - Performant, Stable, and Secure. **Advantages:** - Includes built-in security features and is generally more secure than other platforms. - Highly performant using Apple Silicon chips. **Disadvantages:** - Expensive licensing costs. - Runs on proprietary hardware. - Not typically used for hosting web servers.	**Characteristics:** - Open Source, Stable, Customizable, and Secure. **Advantages:** - No licensing costs. - Can run on many different types of hardware. - Supports many web technologies natively. - Common choice for hosting web servers. **Disadvantages:** - Complex configuration and setup. - Many different distributions and flavors. - Steeper learning curve.	**Characteristics:** - Proprietary, Broad Ecosystem, and Insecure. **Advantages:** - Microsoft tech stack support and integration. - Compatible with legacy systems. **Disadvantages:** - Costly licensing fees. - Resource intensive. - Security concerns.	**Characteristics:** - Portable, Built for Clients, and Relatively cheap. **Advantages:** - Low power consumption. - Highly portable. **Disadvantages:** - Limited resources. - Software and OS limitations. - Not typically used for servers.
Client Side	All clients will be accessing the game via a web browser. Since all client requests will be handled by application code running on the server, the considerations will be the same as the above Server Side section. When the client side code is written, mobile responsiveness should be implemented to ensure a desirable experience for mobile users.	All clients will be accessing the game via a web browser. Since all client requests will be handled by application code running on the server, the considerations will be the same as the above Server Side section. When the client side code is written, mobile responsiveness should be implemented to ensure a desirable experience for mobile users.	All clients will be accessing the game via a web browser. Since all client requests will be handled by application code running on the server, the considerations will be the same as the above Server Side section. When the client side code is written, mobile responsiveness should be implemented to ensure a desirable experience for mobile users.	All clients will be accessing the game via a web browser. Since all client requests will be handled by application code running on the server, the considerations will be the same as the above Server Side section. When the client side code is written, mobile responsiveness should be implemented to ensure a desirable experience for mobile users.
Development Tools	**IDEs:** - VSCode, Xcode, Intellij, Pycharm, Goland, and Neovim. **Languages:** - Java, PHP, Go, Python, JavaScript, HTML, CSS, and SQL. **Frameworks:** - Spring Boot, React, Django, Flask, Angular, Vue. **Considerations:** - The JetBrains IDEs Intellij, Pycharm, and Goland offer community editions and professional editions. If the professional version is used, there is a considerable licensing fee.	**IDEs:** - VSCode, Xcode, Intellij, Pycharm, Goland, and Neovim. **Languages:** - Java, PHP, Go, Python, JavaScript, HTML, CSS, and SQL. **Frameworks:** - Spring Boot, React, Django, Flask, Angular, Vue. **Considerations:** - The JetBrains IDEs Intellij, Pycharm, and Goland offer community editions and professional editions. If the professional version is used, there is a considerable licensing fee.	**IDEs:** - VSCode, Xcode, Intellij, Pycharm, Goland, and Neovim. **Languages:** - Java, PHP, Go, Python, JavaScript, HTML, CSS, and SQL. **Frameworks:** - Spring Boot, React, Django, Flask, Angular, Vue. **Considerations:** - The JetBrains IDEs Intellij, Pycharm, and Goland offer community editions and professional editions. If the professional version is used, there is a considerable licensing fee.	Mobile devices are not a recommended platform for developing software. There are not any full-scale development tools for building web applications.

Recommendations

1. **Operating Platform:** Use Java as the programming language to write the server-side logic and JavaScript/HTML/CSS for all client-side code. These are the most widely used languages and should be familiar to many software developers. The application development framework Spring Boot should be leveraged to build the application. It has a plethora of components for web application development tasks, such as database interaction, authentication, and many others. MacOS, Linux, or Windows should be the operating system used by the developers. They all have excellent support for IDEs and other software development tools. AWS should be leveraged as the cloud hosting platform to enable flexibility, availability, and increased development velocity.

2. **Operating Systems Architectures**: Use Linux as the operating system for the web application server running on an AWS EC2 (a virtual private server) instance. It is the most common OS choice for deploying web applications and is highly customizable and configurable. Ensure that a compatible Java Runtime Environment is installed on the server so that the executable jar produced as the build artifact can run without problems. The developers can write the application using either MacOS, Linux, or Windows, leveraging their IDE of choice. Each of these OSs will meet the requirements.

3. **Storage Management**: The AWS RDS service should be used as the relational database for the application. This will allow all user and game data to be stored, updated, and retrieved in an efficient and scalable manner. AWS S3 should be used for storing all of the images. S3 features unlimited storage and scalability as well as very fast read and write ability for many types of objects.

4. **Memory Management**: Volatile memory (RAM) will be managed by the Java Runtime Environment and the included garbage collector. The application code will be written in a way to efficiently utilize RAM by implementing optimal data structures and software design patterns. The only persistent data that will be stored on the application server's disk will be code and configuration. The database will store all other persistent data like user and game information on its corresponding disk.

5. **Distributed Systems and Networks**: Leverage techniques like AWS EC2 autoscaling to handle any potential server failures. This will enable the number of virtual servers to decrease, and increase based on overall traffic and load on the application. Utilize networking solutions offered by AWS, like Route 53, for routing requests over the public internet to the web application servers. Additionally, a load balancer can be configured to enable efficient routing between backend servers. AWS CloudFront, a popular content delivery network, should be set up to serve all of the images to clients.

6. **Security**: Secure coding techniques will be implemented at every step of development to ensure robust cybersecurity measures. Static vulnerability scanning should be included as part of the build/test/deploy pipeline to proactively identify any outstanding vulnerabilities. Authentication and authorization will be used to mitigate any unauthorized access to customer data. Data

encryption will be implemented for all data at rest and in transit. Only HTTPS connections will be allowed by the server. SSL certificates will be installed and maintained as a further security measure.