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	09/17/23	K. McCracken	Identify initial executive summary and design constraints

Executive Summary

The Gaming Room has requested the development of a web-based game application called "Draw It or Lose It," which is an adaptation of the Android app of the same name. The game involves multiple teams competing to guess a puzzle based on rendered stock drawings. As part of our solution, we will:

- Allow one or more teams to participate.
- Assign multiple players to each team.
- Ensure uniqueness for game and team names.
- Implement unique identifiers to manage instances of games, teams, and players.

Requirements

1. Team and Player Management

Requirement: The application must allow one or more teams to participate, with multiple players assigned to each team.

Description: The application will provide a user-friendly interface for creating and managing teams. Each team can have multiple players, and players can be added or removed from teams as needed. Team names must be unique to prevent naming conflicts.

2. Game and Team Name Uniqueness

Requirement: Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.

Description: When users create a new game or team, the application will check the chosen name's uniqueness. If the name is already in use, the user will be prompted to choose a different, unique name.

3. Unique Identifiers

Requirement: Implement unique identifiers for games, teams, and players to avoid conflicts and confusion.

Description: The application will generate unique identifiers for games, teams, and players. These identifiers will be used internally to manage instances of games, teams, and players and to ensure that there are no conflicts when multiple instances are active simultaneously.

Design Constraints

1. Cross-Platform Compatibility

Constraint: The application must function seamlessly across various web browsers and devices, ensuring accessibility for a wide audience.

Implications:

- Use responsive web design principles to ensure the application adapts to different screen sizes.
- Perform extensive cross-browser testing to identify and address compatibility issues.
- Implement progressive enhancement to provide a basic experience for older or less capable browsers.

2. Scalability

Constraint: The game should be designed to accommodate a growing number of players and teams. This requires a scalable architecture to handle increased loads.

Implications:

- Utilize cloud-based hosting to easily scale resources as the user base grows.
- Implement load balancing to distribute traffic evenly across multiple servers.
- Design the database schema to handle a large volume of user data and game records.

3. Security

Constraint: Security measures should be implemented to protect user data, prevent unauthorized access, and mitigate potential threats such as cheating in the game.

Implications:

- Use HTTPS to encrypt data transmission between the client and server.
- Implement user authentication and authorization mechanisms.
- Employ security best practices to protect against common web vulnerabilities (e.g., XSS, CSRF, SQL injection).
- Monitor and log user activities for security auditing.

4. Real-Time Interaction

Constraint: The game's real-time features, including drawing rendering and player interactions, must be efficient and responsive to provide an engaging user experience.

Implications:

- Utilize WebSocket technology to enable real-time communication between clients and the server.
- Optimize server-side code to minimize latency in drawing rendering and game updates.
- Implement client-side caching and data synchronization for smooth real-time gameplay.

5. Data Integrity

Constraint: Ensure data consistency and integrity, especially in scenarios involving multiple teams and players interacting simultaneously.

Implications:

- Implement database transactions to maintain data consistency during complex operations.
- Use database constraints and validation rules to enforce data integrity.
- Implement conflict resolution mechanisms for scenarios where multiple teams interact with the same game simultaneously.

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

ProgramDriver Class:
- This is a class in our program, but we don't have many details about what it does.

SingletonTester Class:
- Another class in our program, but its purpose is not clear from the diagram.

Entity Class:
- This is like a blueprint for other classes. It might contain common things that other classes will use.

GameService Class:
- This class helps manage game-related stuff. It probably handles things like starting and ending games.

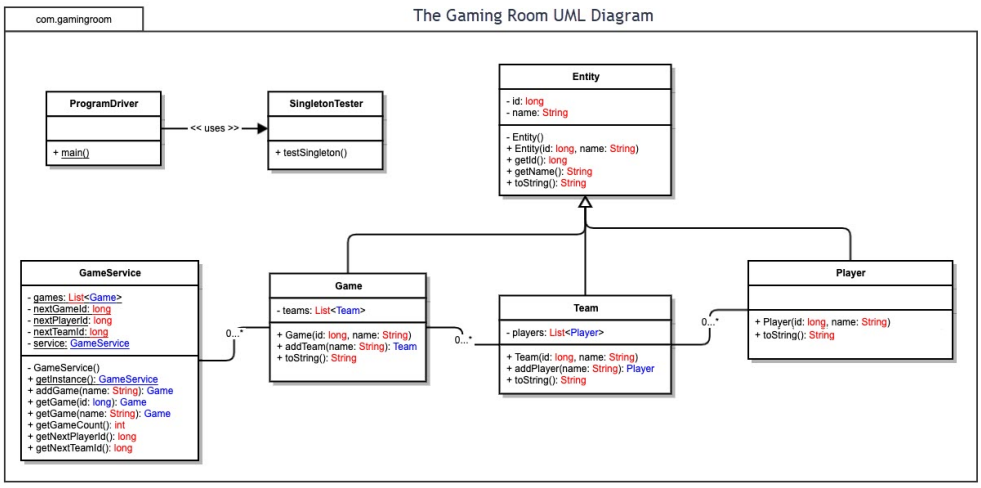
Game Class:
- Represents a single game. If we have multiple games going on, we'd have multiple instances of this class.

Team Class:
- Represents a team within a game. A game can have more than one team, like Team A and Team B.

Player Class:
- Represents individual players in a team, like Player 1, Player 2, etc.

Relationships:
- GameService connects to Game: GameService helps with managing games, so it can be connected to multiple games.
- Game connects to Team: A game can have multiple teams, which is why it's connected this way.
- Team connects to Player: Teams can have multiple players, and this relationship shows that.

This diagram helps us plan how our Java program will be organized and how different parts will work together. It's like drawing a map before you start building something so that you know what goes where.



Evaluation

Development Requirements	Mac	Linux	Windows	Mobile Devices
Server Side	Mac systems are known for their stability and reliability, making them suitable for hosting web-based applications. They offer a Unix-based environment, which is advantageous for web development. However, they may have limited hardware options for hosting large-scale applications.	Linux is a popular choice for server hosting due to its open-source nature, stability, and security. It offers a wide range of server distributions suitable for different application types. However, it might require more expertise to configure and maintain compared to Mac or Windows.	Windows servers are widely used and offer good support for .NET and ASP.NET applications. They are known for their user-friendly interface but may not be as cost-effective as Linux for hosting. Windows servers may require more frequent updates.	Mobile devices are not typically used as server hosts for web-based software applications due to limitations in terms of processing power, memory, and network connectivity. They are more suited for client-side use.
Client Side	Supporting Mac clients may require additional considerations in terms of cost and time due to the need for Mac-specific development and testing. Expertise in macOS development may also be necessary.	Supporting Linux clients can be cost-effective as Linux is open source. However, it may require extra time for testing and adaptation to different Linux distributions. Linux expertise may be needed.	Windows clients are widely used, and development for Windows is well-supported. It may require less time and expertise to develop for Windows clients.	Supporting mobile devices as clients can be challenging due to the variety of platforms (iOS, Android) and screen sizes. Developing for mobile may require additional resources and expertise.
Development Tools	For deploying on Mac, relevant programming languages include Swift and Objective-C for native macOS applications. Integrated Development Environments (IDEs) like Xcode are commonly	Linux supports a wide range of programming languages like Python, PHP, and JavaScript. Popular tools include text editors like VSCode and JetBrains IDEs.	For Windows, programming languages like C# for .NET applications are common. Visual Studio is a popular IDE.	Developing for mobile devices requires knowledge of platform-specific languages like Swift (iOS) and Java/Kotlin (Android). IDEs like Xcode (iOS) and Android Studio (Android) are used.

Recommendations

1. Operating Platform: I recommend using a cross-platform operating system such as Linux. Linux offers the advantage of being open-source, which can lead to cost savings. It is known for its stability, scalability, and strong support for web-based applications. Additionally, Linux is compatible with various computing environments, making it suitable for expanding Draw It or Lose It.

2. Operating Systems Architectures: Linux comes in various distributions, each with its own architecture, but broadly, Linux follows a monolithic kernel architecture. This means that the kernel manages both core operating system functions and device drivers. This architecture offers efficiency and robustness for server-side applications like Draw It or Lose It.

3. Storage Management: For storage management, Linux offers a wide range of options, including traditional file systems (e.g., Ext4), networked file systems (e.g., NFS), and distributed file systems (e.g., HDFS). The choice would depend on the scalability and data storage needs of Draw It or Lose It. A combination of local storage for game data and networked storage for user accounts and leaderboards could be a viable solution.

4. Memory Management: Linux uses sophisticated memory management techniques, including virtual memory, paging, and process isolation. These techniques ensure efficient memory allocation and utilization. For Draw It or Lose It, this means that the application can run efficiently without memory leaks or resource contention issues.

5. Distributed Systems and Networks: To enable communication between various platforms, a microservices architecture can be adopted. Draw It or Lose It can be broken down into smaller, independent services that communicate over RESTful APIs or message queues. These services can run on different servers or containers and can be hosted on multiple platforms. Docker containers and Kubernetes for orchestration can help manage the distributed components effectively. Additionally, a reliable network infrastructure with load balancing and failover mechanisms should be in place to ensure connectivity and handle outages.

6. Security: Security is a top priority for The Gaming Room. Linux provides robust security features, including user and group permissions, firewall (iptables), and SELinux/ AppArmor for process isolation. Data encryption, both in transit (using HTTPS) and at rest (using encrypted file systems or databases), should be implemented to protect user information. Regular security updates and patch management are crucial to keep the system secure. Additionally, user authentication and authorization mechanisms should be implemented, possibly using OAuth or OpenID Connect, to ensure user protection.