CS 230 Project One Game App

CS 230 Project Software Design Template

Version 1.0

Table of Contents

Document Revision History

Version Date Author Comments

1.0 01/26/25 Jake Gagnon CS 230 Project One Game App.

The Gaming Room wants to develop a web-based version of their game, "Draw It or Lose It," to support multiple platforms.

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 wants to develop a web-based version of their game, "Draw It or Lose It," to support multiple platforms. Currently, the game is only available on Android, and they aim to extend its reach to web-based platforms while ensuring features such as team-based play, unique names for games, teams, and players, and efficient memory management with a singleton design. The proposed solution includes a structured design based on object-oriented principles to ensure scalability, maintainability, and compliance with client requirements.

Requirements

< Please note: While this section is not being assessed, it will support your outline of the design constraints below. In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>

Design Constraints

Constraints and Implications:

⦁ Web-Based Distributed Environment:

⦁ The game must function seamlessly across different devices and browsers, which may require adherence to web standards and the use of frameworks.

⦁ Implication: Development complexity increases, and cross-platform testing is necessary.

⦁ Singleton Design Pattern for GameService:

⦁ Only one instance of the GameService class is allowed in memory to manage games, teams, and players.

⦁ Implication: Simplifies game state management but requires careful handling to avoid memory leaks or race conditions in a multi-threaded environment.

⦁ Unique Identifiers for Entities:

⦁ Games, teams, and players must have unique names and IDs to prevent conflicts.

⦁ Implication: Requires robust logic to validate and enforce uniqueness, increasing the computational overhead when creating new entities.

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 provided UML diagram demonstrates a clear object-oriented design:

⦁ Entity Class:

⦁ Acts as a base class for Game, Team, and Player.

⦁ Common attributes (id, name) and methods (getId(), getName()) are centralized to reduce redundancy and promote reusability.

⦁ Demonstrates inheritance, allowing subclasses to inherit shared properties.

⦁ GameService Class:

⦁ Manages the lifecycle of games, teams, and players using lists.

⦁ Implements the singleton pattern, ensuring only one GameService instance exists to manage global game state.

⦁ Relationships:

⦁ GameService to Game (0...\*): One GameService manages multiple Game objects.

⦁ Game to Team (0...\*): A game contains multiple teams.

⦁ Team to Player (0...\*): A team contains multiple players.

⦁ Polymorphism:

⦁ Methods like addGame(name) in GameService demonstrate polymorphism, allowing flexibility in handling entities dynamically.

⦁ Encapsulation:

⦁ Private fields (id, name, etc.) ensure internal state is protected and accessed through public methods (getId(), getName()).

⦁ Implement HTTPS to secure communication between clients and the server.

⦁ Use authentication mechanisms (e.g., OAuth 2.0) to protect user accounts and data.

⦁ Sanitize user input to prevent injection attacks.

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 MacOS can host a web-based software application effectively with its UNIX-based architecture. It is stable, supports many server technologies, and has tools like Apache, Nginx, and Docker. However, it is more expensive than Linux-based servers and lacks the widespread enterprise support Linux offers. Linux is the preferred platform for web-based hosting due to its cost-effectiveness, scalability, and flexibility. It is open-source, secure, and widely used in enterprise settings. Weaknesses include a steeper learning curve for administrators unfamiliar with the command line. Windows Server offers strong integration with enterprise software like Microsoft SQL Server and .NET Core. However, it is expensive and resource-intensive compared to Linux. Windows servers are often less preferred for high-traffic web hosting due to licensing costs. Mobile devices are not suitable for hosting web applications directly due to limited resources and connectivity challenges. However, mobile devices rely on back-end servers for app communication.

Client Side Development for Mac requires familiarity with Apple’s frameworks (e.g., Swift, Objective-C). Costs include macOS hardware, which is premium-priced. Development is time-consuming if cross-platform tools are not used. Development for Linux desktops has a smaller market share, and cross-platform tools (e.g., Electron, Flutter) are often required. Time and cost are lower due to open-source resources. Expertise in Linux distributions is required for development. Windows development is widely supported, and most desktop users are on Windows. Tools like Visual Studio for .NET make development efficient, though licensing costs for enterprise tools are higher. Expertise in Windows development is widely available. Mobile app development requires expertise in Android (Java/Kotlin) and iOS (Swift/Objective-C). Cross-platform tools like Flutter, React Native, or Xamarin reduce cost and time but may not provide native performance.

Development Tools Common tools include Xcode, IntelliJ IDEA, or Eclipse. Programming languages include Swift, Java, and Python for back-end and front-end web-based applications. Languages like Python, Java, PHP, and JavaScript are commonly used. IDEs such as Visual Studio Code, Eclipse, and PyCharm work well on Linux. Docker and containerization are widely supported. Languages like C#, .NET, JavaScript, and Python are supported. Tools include Visual Studio, Visual Studio Code, and SQL Server Management Studio. Languages include Swift, Kotlin, Java, and Dart. IDEs include Android Studio, Xcode, and Visual Studio for Xamarin or React Native development.

Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

⦁ Operating Platform: Use Linux as the operating platform due to its scalability, cost-efficiency, and robust server-side capabilities for hosting web applications.

⦁ Operating Systems Architectures: Adopt a 64-bit architecture to handle high-performance computing, support larger memory, and improve scalability.

⦁

⦁ Storage Management: Implement a relational database system (e.g., MySQL or PostgreSQL) to store game, team, and player data. Use unique constraints and indexing to enforce uniqueness and optimize data retrieval.

⦁ Memory Management: Leverage garbage collection (available in Java and most modern web frameworks) to handle memory efficiently. Use caching mechanisms for frequently accessed data, such as game states.

⦁ Distributed Systems and Networks: Use RESTful APIs to facilitate communication between the client and server. Employ a load balancer and CDN to ensure high availability and performance for a distributed environment.

⦁ Security: Implement HTTPS to secure communication between clients and the server.

⦁ Use authentication mechanisms (e.g., OAuth 2.0) to protect user accounts and data.

⦁ Sanitize user input to prevent injection attacks.