

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.2

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.2 | 02-21-23 | Brian H. Argenti | N/A |

**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](#_sbfa50wo7nsh)

Our company has recently teamed with The Gaming Room to develop a web-based game based on their current IP *Draw It or Lose It,* an Android based app. The client requests the game needs to be able to run on multiple platforms and multiplayer capable of handling multiple teams and players on a single server. However, our client is unsure about how to set up the environment that will allow a browser-based version of their game operate in this way. Creative Technology Solutions will attempt to solve this problem by building a Java program capable of creating single unique instance of the game that will be able to host multiple teams and players, each with unique IDs.

## 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](#_2et92p0)

* We need to be able to create a version of the game that is runnable on all types of browsers may require the need to code in HTML5 or JavaScript.
* There needs to be only one instance of the game running at any given time that multiple people can access and join.
* The game must be able to render a drawing within 30 seconds at a steady rate of x pixels per second and accept input from players to determine the correct answer.
* There must be an implementation of a 15 second timer for the opposing team to guess the answer.

## [System Architecture View](#_ilbxbyevv6b6)

N/A

## [Domain Model](#_8h2ehzxfam4o)

In the UML Diagram below is a Singleton Design Pattern where the Entity class is inherited by the Game, Team and Player classes. The Entity class protects the id and game attributes by abstracting them and only allowing access through the public method calls of Game(), Team(), and Player(). This also shows how the program uses polymorphism to be used by all of the subclasses, increasing efficiency. The GameService class creates an instance of the game with a unique ID so any other calls to the game will not create another instance in the memory. GameService encapsulates the data through the use of accessors.

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## [Evaluation](#_2o15spng8stw)

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** | Server side development will be completed by writing RESTful APIs in the Eclipse IDE using a Maven build. This is accomplished using a Java framework known as Dropwizard, with a few tools: Jetty (an HTTP library), Jersey (a reference implementation framework) and Jackson (a JSON (JavaScript Object Notation) library for Java). This will ensure any web browser used will be able to successfully complete the call made from the client-side allowing users of various platforms to all be hosted at the same time while a single instance of the game is running. | | | |
| **Client Side** | MacOS has the Safari browser and is rendered using the WebKit engine which works on all apple devices. Because we are using REST APIs, cost should be minimal. We will need time to develop the program and people experienced in writing Java and using REST to develop web applications. | Linux can use any variety of browsers but most “distros” used to browse the web should be able to handle website rendering. The RESTful framework should keep costs down by not having to re-write the code for multiple browser types. Time and experience levels should be minimally affected as we should only need to write the program once. | Windows by Microsoft can also run various browsers however the native browser is Edge which is based off of Chromium (an open source web browser) and loosely imitates Google’s Chrome browser, for good reason. Because we are using RESTful APIs, there should be no cost, time or special expertise needed since all browsers should be able to make the call to our server and the call should be able to be completed. | Mobile devices not running MacOS should probably be running some version of Chromium (Chrome, Edge, Vivaldi and Samsung’s native browser all run Chromium). With most browsers capable of handling RESTful calls to the server side, there should be little change in outcome besides maybe adding a perspective change for mobile users to better accommodate the perspective which could incur more costs and time in development. |
| **Development Tools** | All development should be on the server side so there should not need to be any development required for the client side. This means the Eclipse IDE using Java to build the code and the Dropwizard framework for developing RESTful APIs should be all that is required for deploying the software on Mac. | Like the Mac, development will be done in Eclipse and use Java. Using the RESTful APIs ensure the code only needs to be written once. Only esoteric “distros” may be affected but it would be a super minority that would not affect overall deployment of the software on Linux. | Java on Eclipse would be the language and IDE used for windows just like the others. This is the beauty of creating projects using RESTful APIs. JSON would also be used in building these programs which should be included in all considerations within this row, not just Windows. | Eclipse has an IDE for mobile as well, complete with Java development tools so all platforms can and should be able run on the client side the application that was developed on the server side with minimal issues. |

## 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 would recommend Windows because of its ubiquity and suite of tools to help indie developers and larger operations meaning there is scalability for the event there is a need. Also, Microsoft’s push to go open source has made it more friendly to developers which should allow The Gaming Room to expand their game to other computing environments.
2. **Operating Systems Architectures**: With Microsoft Azure there are many pre-fabricated architectures to choose from depending on what the program needs to do. For our purposes we could use a single-page application (SPA) with a majority of the UI logic performed in the web browser and using an API to communicate with the server. An SPA will allow the client-side to take actions without needing to reload the page so as the image renders, teams would be able to make guesses in real-time.

1. **Storage Management**: Because of Windows embrace of open sourcing, the ability to take advantage of other storage options exists without completely relying on Microsoft. For that, I would recommend Amazon S3, which is an object storing service that has the ability to store text and pictures, the kind of data we will mostly be handling with our project.
2. **Memory Management**: Everything will be stored in the S3 bucket and the use of REST API’s will be used to call for the items. The amount of memory used to load the pictures and then render them at a steady rate should be minimal. The base of the program will be written in Java which supports garbage collection. This essentially frees up the memory when the program recognizes the data in memory is no longer useful. This should keep memory allocation relatively free.
3. **Distributed Systems and Networks**: I have previously mentioned the use of an API to communicate with data on the server-side. By using a RESTful API, multiple platforms can make the call to the data from the client-side and allow multiple operating systems to have the ability to communicate with each other. AWS S3 services can handle a surprisingly high amount of requests per second, with tricks to scale the amount up as high as one might need using “prefixes”.
4. **Security**: In order to secure data, Microsoft employs encryption tactics which they refer to as Encryption at Rest. It is the secure encoding of data that uses symmetric encryption to encrypt and decrypt data with the use of a secure key that is stored elsewhere with identity based access control and audit policies. As for creating a secure app, a username and unique password can be implemented through the use of Java and an API to set up an Authenticator. No matter which platform is used, the program will make the call for the authentication and will prompt the player for a username and password.