

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.2

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/11/2022 | Austin Henley | Created Executive Summary, Design Constraints and Domain model |
| 1.1 | 11/26/2022 | Austin Henley | Created Evaluation of Operating Systems |
| 1.2 | 12/10/2022 | Austin Henley | Completed Recommendations |

## 

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room currently has an android-based game called Draw It or Lose It, a game loosely similar to a 1980s game *Win, Lose or Draw* in which teams compete to guess what is drawn. They are wanting to port it into a web-based game that serves multiple platforms and operating systems.

## [Design Constraints](#_2et92p0)

* The game will need to be fully cross-platform. Currently the game is only available on android, and in making it compatible across multiple platforms, the game will have to be translated into a web application.
* The online application will need to match the user interface of the android application. This will keep consistency across versions of the game, while also allowing familiarity for users.
* There will need to be an ability to have one or more teams involved in each game, while also having multiple players across teams. The web application will need the ability to assess when there are duplicates and conflicts due to multiple people, while also maintaining memory usage due to unique identifiers for each person.
* Only one instance of the game can exist at a time. Object-oriented programming principles will need to be used here to ensure that there is only one instance of the game at one time.

## [System Architecture View](#_ilbxbyevv6b6)

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](#_8h2ehzxfam4o)

In the UML class diagram, we can see the importance of the Entity class. The Entity class acts as a base class for the Player, Team, and Game class, as they inherit all the methods of Entity. Throughout the program, the methods in Entity, used through each of its subclasses, allow us to have multiple teams with unique identifiers and names (seen with the variables “id” and “name”). Each subclass then has a relationship with one another. Since Teams consist of players, the Team class has a method to create a list of players. With on of the technical requirements being the need foy one or more teams, the Game class has a list of teams included within the class. The GameService class is a major necessity for the application since it houses the functionality of swapping games and creating the unique identifiers for each object. To ensure that there is only one instance, the GameService class has a singleton pattern built into it to only have one GameService object that is created privately at the beginning of running the program. The ProgramDriver class will be what is ran at the start of the program since it houses our main method. This is where it uses the SingletonTester to ensure there is only one instance of our game service.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | | **Mobile Devices** |
| --- | --- | --- | --- | --- | --- |
| **Server Side** | The Mac ecosystem is proprietary to Apple, and this comes with its advantages and disadvantages. The cost upfront is higher due to the hardware costs for Mac computers, software, and licenses. This comes with superior reliability as Apple provides heavy support for their products and provides updates when bugs are found. The administration of the server is user-friendly due to the GUI. | Linux is an open-source software, so the cost overall can be much less. Linux servers can be run across many different types of hardware, and there are many free options when it comes to software. Ease of use is a challenge here for the administrator, and there is a learning curve for those inexperienced. With that learning curve though, Linux’s open-source nature allows freedom to make changes but requires security to be considered and implemented. | The Windows ecosystem is also proprietary to Microsoft. Windows servers support many third-party applications and add-ons. There is also consistent support from Microsoft through system updates. Cost is still higher with Windows due to licensing costs, and the hardware costs, although less than Mac, still can be higher than Linux. Windows is user-friendly, providing a graphical interface to control the server. | | Mobile devices can be used for web servers, but typically are not used due to a lack of resources. When hosting a web application that will consist of multiple users and moving parts, it is important to have the power behind it to handle that usage. Typically, in a mobile device situation, the cloud would be used to house server data and could provide an opportunity for The Gaming Room app but would be more expensive month-to-month, as there would be a cost involved with paying another company to house the server components. |
| **Client Side** | With The Gaming Room being a web app, compatibility will have to be considered with major web browsers on macOS, such as Safari and Chrome. The consideration would come down to optimizing media types for Mac, which supports the same types as Linux and Windows. The cost would be built into creating the web app to begin with. | Linux’s major browsers are Firefox and Chromium, which support HTML web apps. Consideration may be needed for major Linux distributions, as not all clients will be using the same version of Linux. If the web app is optimized for browser usage during development, the cost and time should be consistent with the other desktop Oss’. | Window’s major browsers include Firefox and Chrome, with Microsoft making a major push for their Edge browser. All these browsers have consistent support for HTML web apps, and if the app is optimized for browser usage during development, cost and time should be consistent with other desktop OSs’. | | Clients connected from mobile devices will be visiting from several different screen sizes and input types. Extra time will be added to development to allow further optimization for these varying devices. Media would need to be optimized to fit the smaller screen and still be presentable, and input through the touch screen would need to be allowed. iOS has Chrome and Safari, while Android has Chrome and Samsung's proprietary internet browsers, and all are compatible with desktop sites, but optimization for mobile sites would need to be added. |
| **Development Tools** | To build the front-facing portion of the web app, the most useful programming languages would be:  HTML – To create the structure of the app  CSS – To style the website and  JavaScript – To allow dynamic behavior necessary for actions to be performed  The backend of the site could be built using Java, Python, or any of the other modern programming languages.  HTML, CSS, and JavaScript can be written and edited through Visual Studio Code, while a different IDE will be needed to write Java or Python. For Java, Eclipse can be used and PyCharm can be used for Python.  If we were to consider creating an app for the app store, the development would need to be shifted to Apple’s swift programming language and the xCode IDE would need to be used. There is also a 99/year license for distribution on the app store. | To build the front-facing portion of the web app, the most useful programming languages would be:  HTML – To create the structure of the app  CSS – To style the website and  JavaScript – To allow dynamic behavior necessary for actions to be performed  The backend of the site could be built using Java, Python, or any of the other modern programming languages.  Linux offers many alternative tools for writing code due to open-source flexibility. Fortunately, the same tools as both Windows and Mac are available on Linux as well, so Visual Studio Code would be used for HTML, CSS, and JS while Eclipse and PyCharm would be used for Java and Python. | | To build the front facing portion of the web app, the most useful programming languages would be:  HTML – To create the structure of the app  CSS – To style the website and  JavaScript – To allow dynamic behavior necessary for actions to be performed  The backend of the site could be built using Java, Python, or any of the other modern programming languages.  Windows has many of the same tools that are available across other OSs'. Visual Studio Code would be used for HTML, CSS, and JS while Eclipse and PyCharm would be used for Java and Python. | Since this app is being developed for the web, separate tools would not be necessary for mobile devices to be compatible.  If mobile standalone apps were to be entertained, other considerations would need to be made.  Creating an app for iOS would require the use of the Swift programming language and xCode IDE on macOS.  On Android, the Android Studio IDE would need to be used. Java and Kotlin are the official languages of Android development. |

## Recommendations

1. **Operating Platform**: The focus of this project is the product experience and scalability of the application. To focus more on the software itself, I would suggest the use of a cloud platform such as Amazon Web Services (AWS) Elastic Beanstalk. Elastic Beanstalk is more of an orchestration service but utilizes many of Amazon's services to run the application within a Linux OS. There is a monthly cost associated with running through Elastic Beanstalk, but the cost allows the focus of the team to be on customer experience, and reduces the overhead associated with maintaining a scalable web application.
2. **Operating Systems Architectures**: Since we will be utilizing a cloud-based platform, many systems are working together to host our web application. At the core of the system will be a system instance running within Amazon's EC2 Linux platform, hosting the main instance of our application. This will allow a stable environment to run within a UNIX-based operating system.
3. **Storage Management**: For storing our images and information, we would be utilizing an AWS storage solution since it integrates with our platform. Amazon S3 object storage would be a good option here, as it is a stable and secure storage solution, that allows scalability. It provides several tools to be able to manage what is stored and is completely private to the resource owner.
4. **Memory Management**: Content delivery is an important factor to consider. With using a cloud-based platform, the biggest consideration will be how to get images for Draw It or Lose It to the customers synchronously and fast. As it works in conjunction with the rest of the platform, the Amazon CloudFront content delivery network will be our best choice here. Amazon’s network is globally dispersed and would allow cached images closer to the customers. Along with CloudFront, we could utilize ImageKit, an image optimization service, to optimize the photos as they are requested by the user. ImageKit would work in conjunction with CloudFront to cache images closer to customers and bring down latency and loading times.
5. **Distributed Systems and Networks**: Many services will be working together to create a functional web application. Cloud Services we would be utilizing:

* Amazon Elastic Beanstalk – Deployment of the application
* Amazon EC2 – Utilized through Elastic Beanstalk to run system instance
* Amazon S3 – Storage solution for application images and information
* Amazon CloudFront – Edge caching for faster content delivery to customers
* AWS WAF – Security built-in to CloudFront to help protect against malicious traffic
* AWS Shield – Create safeguards against DDoS attacks

There is always a possibility of failure when using a cloud-based infrastructure, but we would utilize testing and automation to ensure reliable communication across platforms. More specifically, we can use key performance indicators to monitor performance from a user standpoint. If a slowdown were to occur, we would have an automated recovery process to repair the failure and find a solution. Fortunately, with a cloud-based platform, it is easier to monitor demand and use, and resource availability can be scaled for user demand.

1. **Security**: There are tools built into the AWS platform to protect from certain security risks. We would make use of AWS WAF to manage malicious traffic against our CloudFront CDN. This would act as a firewall against any type of malicious activity searching for user data. This would be used in conjunction with AWS Shield, which automatically detects and mitigates network-level DDoS attacks. Amazon’s AWS platform also has secure access control built into their networks to protect against outside interaction. Due to the wide variety of web browsers on the market, security efforts will be built on the server side.

Within the application itself, Authentication would be built into the stack. The proper authority would be required to make any changes through the API. We would utilize role-based access on the client side.