

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## 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 | 7/18/2021 | Chris Austin | Initial Requirements |
| 2.0 | 8/2/2021 | Chris Austin | Updated Evaluations |
| 3.0 | 8/15/2021 | Chris Austin | Final Recommendations |

**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)

The Gaming Room wants to reach many users across browser and devices with the Draw It or Lose It application. To achieve this goal, Draw It or Lose It should be implemented with proven commercial grade tools and platforms including Java, Linux and a RDBMS. The user interface will be implemented using modern web technologies including Javascript and CSS frameworks.

## [Design Constraints](#_2et92p0)

Draw It or Lose It is to be delivered in a modern, distributed, web environment. Along with the application’s requirements, this creates the following design constraints.

**Cross Platform:**

To reach users on any platform tools and technology used to develop Draw It or Lose It must not limit a user’s access to the game. Web-browser on all major desktop and mobile platform should have a consistent, modern and pleasant experience. This implies using cross platform development tools and approaches. Heavy emphasis is placed on using tools that reach as many platforms as possible without needing to produce platform or browser specific builds or code.

**Use Cross Browser Tooling:**

The customers main goal is to reach users wants user to access the game via common web technologies. The most straight forward method of doing this is to use tools and technology that produce cross browser applications. Modern cross browser tools produce JavaScript and CSS that renders natively on iOS, Android, Windows and MacOS browsers.

**Highly Available / Scalable:**

Web-based games can deliver many thousands of users in a short period of time. Applications that are not able to scale to these number of users can develop poor reputations and lose users quickly. So, we must ensure that our design takes into account scalability and availability. This means that any services we deliver must be able to be load balanced and scaled with minimal friction. This implies non-sticky sessions for users and a runtime environment that can scale both vertically and horizontally. Further, hosting, and infrastructure costs should be kept at a minimum by avoiding expensive licensing for server side technologies.

## [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)

The core model used for user interaction in this application are Game, Team and, Player. Each of the entities inherit from the base model Entity. By extending Entity the models Game, Team and, Player instances have a polymorphic approach to providing identifiers, names and, string representations of the internal data. The internal data of these models is encapsulated by using private member variables.

A Game is composed of (has) zero or more Teams and, a Team is composed of (has) zero more Players. By using Entity as a base, Game, Team and, Player can ensure unique names and Ids.

The GameService model provides a composite approach for users to interact with the entity models Game, Team and Player. An instance of the GameService has one or more games. Additionally, the GameService model is a Singleton so, only a single instance of this model can exist in memory at a time. Also, GameService’s accessors and mutators that provide functionality for adding, getting and iterating Games, Teams and Players while encapsulating the underlying implementations.

Finally, the ProgramDriver Model wraps the SingletonTester entity to provide a way to launch the GameService entity.

****

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | MacOs has a solid Unix / Posix core. However it is primarily a desktop OS and require additional resource and workarounds to be used as a production server. Cloud hosting of MacOS is limited and pricey. | Server versions of Linux have great networking stacks and tools for running headless services. Depending on the distro and host there are cheap to zero licensing cost. Additionally Linux is the de facto choice in public and private cloud environments. | Windows Server via .net has decent hosting capabilities. However, it lacks the adoption in cloud environments and is finicky with any tools and runtimes except those provided by Microsoft. Windows Server licensing also quite expensive. | Mobile devices do not meet basic requirements for hosting services. |
| Server Side  Deployment | MacOS doesn’t provide any specific Server-Side deployment tools. However, standard Linux/Unix + Posix tools are available.  Salt/Puppet/Ansible are all available for MacOS  *But* deploying with Docker/Containers is not recommended because Docker requires a VM to run on MacOS. | Linux doesn’t provide any specific Server-Side deployment tools. However, standard Linux/Unix + Posix tools are available.  Salt/Puppet/Ansible are all available for Linux,  Docker / Containers is a strong option for deployment. Docker runs natively with Linux. | Windows provides .net specific deployment tools.  Servers can be packaged as windows installers.  Standard Linux/Unix + Posix tools are **not** natively available.  Salt/Puppet/Ansible are all available for Windows.  Windows native docker support is limited and not mature. | Mobile devices do not meet the basic requirements of hosting a service. |
| Server-Side  Licensing  Costs | MacOS licensing requires purchasing Apple hardware.  MacOS in cloud computing environments requires expensive MacOS hardware / VMs. | Linux has no inherit licensing costs.  Commodity VMs and servers are available for cloud hosting and self hosting. | Windows can have significant licensing costs depending on hosting options.  - In-house hosting requires purchasing and maintaining Windows licenses.  - Cloud hosting prices for Windows servers and VMs are typically much more than Linux equivalents. | N/A |
| **Client Side** | Developing Mac native applications requires some expertise with Apple’s XCode tools. This requires purchasing MacOS systems and developers with this specialized knowledge.  However, web browser based application can be accessed via a web browser on MacOS. | Desktop applications in Linux are typically developed in Java Swing, QT and, GTK. Developing a native application for Linux would include the cost and time for using these tools while not reaching the same number of users running MacOS and Windows. Like MacOS, Linux users can be reached via browser based application. | Windows is the dominant desktop platform. Developing native apps can be done with .net, C++ and other tools. Developers are plentiful while tooling is well know and affordable.  Web browser based applications can also be used with this platform. | Developing for Mobile Devices can be done with native iOS and Android tools. Additionally, cross platform tools can be used to support iOS and Android with a single codebase and minimal OS specific changes.  Web browser based applications can also be used. However, care must be taken to ensure consistent behavior and appearance across device types and screen sizes. |
| **Client Side** |  |  |  |  |
| **Development Tools** | For consistent browser support, modern CSS and javascript tools should be employed.  The reccomended Javascript / CSS tools are opensource and license free. | For consistent browser support, modern CSS and javascript tools should be employed.  Server side development with Linux can be done with Java, Python, C, C++ and many other languages. IDEs and environments are varied and depend on the developer’s preferences. | For consistent browser support, modern CSS and javascript tools should be employed.  Mainstream windows Server development employs Visual Studio (and VS Code) along with .net languages like C#. | For consistent browser support, modern CSS and javascript tools should be employed. |
|  |  |  |  |  |

## 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 (Server)**:

**Host OS:** It is recommended that Gaming Room host Draw It or Lose It services on Linux. This is because Linux supplies world-class networking, memory management, storage management and, security features.

**Physical/Runtime Environment:** It is recommended that Gaming Room use AWS or another public cloud provider for hosting servers. This provides dynamic load-balancing, flexible and fault tolerant storage, and the ability to scale services based on demand.

**Services:** Back-end services should be developed in Java. Java and tooling is mature has many out-of-the box features that can speed development. The app will be delivered via a browser-based application to reach the most devices without needing to adopt native tools. This is provided by using modern Javascript and CSS frameworks.

1. **Development Costs:** 
   1. **Front End / Client-Side:** To minimizedevelopment costs *Modern, Cross Platform and, Performant* web-development technologies should be employed. Component development tools like Vue.js and React.js along with cross browser CSS tools like PostCSS enable a single code base for all major OSs and browsers.
   2. **Server-Side / Back End:** Development time should be minimized by using comidity / well understood approaches for providing data to all client types. This means using a RESTful architecture with frameworks such as Spring Boot , DropWizzard, Django. These frameworks provide consistent and proven tools for the following:
      1. Managing users.
      2. Interacting with databases
      3. Servicing HTTPs requests
      4. Providing RESTful resources to clients
      5. Securing web and static resources.
      6. Managing the runtime features and performance of the service.
      7. Emitting errors and metrics to help development and operations find and squash bugs.
   3. **Development Tools licensing costs**:
      1. Market leading Client-Side cross platform development tools for browsers are open source and free of licensing costs. Some tools offer premium support that can be purchased as needed.
      2. IDEs / development environments can have some licensing costs. However, free to use and Open Source tools are available if the client does not want to purchases these tools.
   4. **Team Composition:** Two basic skill sets must be filled to build performant, distributed, web-based applications. These are
      1. Back-end / server-side development.
      2. Front-end / client-side development.

Due to wanting to develop a highly responsive product, it will be recommended to staff a cross functional development team with some developers who are stronger in server-side development and others who are stronger in client-side development. This will allow the client-side developers to quickly build a high-quality cross platform / cross browser user interface. At the same time, the developers who are strong in server-side development can focus on quickly building a high-quality, performant and, secure application server.

1. **Operating Systems Architectures**: Hosting the back-end services on Linux (or Linux based containers) allows Draw It or Lose It to be hosted in-hose or in public cloud environments. Linux networking is proven reliable and robust. Linux servers typically lack a GUI but have powerful CLIs and scripting environments. Java and the JVM provide a battle tested runtime environment with access to a large selection of core and third-party libraries. The JVM requires ample RAM to avoid unnecessary garbage collection so, our Linux host’s will require 8GB plus of memory.

The user interface is hosted in the web-browser itself via Javascript.

1. **Storage Management**:

Public and Private Cloud Platforms provide offerings well suited for storing application files, application assets and persistent user data. This includes object storage, expandable OS/Volume Storage and, content delivery networks. Because it is recommended to use a Public Cloud as the runtime environment for the Linux hosts / application servers, it is also recommended that “Lose It or Draw It” uses these storage features when and where it is appropriate.

**Some Quick Definitions:**

Object Storage: Object storage is a class of flexible storage that can be used to store unstructured data and files that do not belong in databases. Object storage providers often offer flexible / near-unlimited capacity with features like archiving, versioning and, integration with CDS.

OS/Volume Storage: Volume storage is a storage class that is attached to the host operating system. Volume is attached to hosts and provides fast access in a native manner. Volume storage is typically offers high performance using SSD (or better) based storage.

Content Delivery Network (CDN): A CDN provides fast access to web application content by storing a web application’s assets (application code files and images) in distributed locations near the end users. This improves how quickly applications are loaded and updated in the web browser.

**Client Application Code:** JavaScript application code is bundled into downloadable and minified files that are loaded by the web browser when a user visits the application’s URL. This minified code should be stored using Object Storage and made available via a CDN. This allows “Lose It Or Draw It” to provide a responsive and performant browser-based application to the end-users.

**Client Application Assets**: Like the Client Application Code, Object Storage and a CDN should be used to quickly deliver images and drawing to the end users.

**Server Application Code:** Server Application code should be stored in highly performant volume storage. This volume storage will be attached to the Linux Servers running the server application. This ensures that the server is not incurring latency due to IO operations.

**User Data**: For storing persistent Player, Team and Game state a relation database should be employed. RDBMSs like Postgres, SqlServer and MySql provide high performance and reliability while being well understood and able to run on nearly any self-hosted or cloud environment.

1. **Memory Management**:

**Host OS:** Like other mainstream OS’s Linux uses RAM to deliver and route information to and from the CPU. Additionally, modern CPU architecture employs L1, L2 and L3 caches to store common instructions which speeds CPU operations and reduces the need to access ram or virtual memory. The Linux application servers should be well equipped to run our Java / JVM server application in order to further reduce any need for virtual memory due to is poor performance compared to RAM. Finally, because our Linux servers will be using SSD or better class of storage, if the need to page / access virtual memory arises, the incurred IO cost should be minimized.

**Server Application:** The JVM creates runtime memory in a heap. Memory is managed and de-allocated via Garbage Collection. During program execution, applications running the JVM access the heap to allocate and reference memory. When the JVM detects memory is low the garbage collector is invoked.

**Client:** The Client in this case is a web browser running a JavaScript application. A browser allocates memory by requesting it from the host OS as it processes the application code. In cases on devices where RAM is limited, the browser will cache the application code and state to disk. The browser will release the RAM for a running application when the user closes the browser or closes the application.

1. **Distributed Systems and Networks**:

Ensuring seamless user interaction in a distributed environment requires a stateless client-side approach and server-side. In this approach, user sessions are not tied to a server instance. Services are kept available via load-balancing and service discover. With this approach, clients present a user’s credentials via a token and are validated on every request. User state is queried and updated in the RDBMS. In the case of network partitions or outages, services will attempt to retry messages and the load balancer should route messages to available servers.

**Communication via various platforms:** “Lose It or Draw It” will require that major Desktop and major Mobile device users are able to participate with each other in games while being able to communicate easily with the server components. To accomplish this, clients and servers will use TCP over IP to provide reliable communication. Additional, Clients will communicate with Servers via REST over HTTP. HTTP is a TCP protocol used by browsers to provide hyper media and REST is an Architectural approach from making application resources available for clients. This ensures a common and simple approach to communication.

**Dependencies:** In a distributed system Clients depend on Servers for establishing and maintain application state. Additionally, Servers often depend on other Servers. In “Draw It or Lose It”, Clients depend on the RESTful application Servers and the application Servers depend on the Database Server.

**Resiliency:** In order to minimize outages, Application Servers will not be a single point of failure. At least two servers will be up at any time and traffic will be routed to them via a Load Balancer. This provides scalability and ensures that if a single Application Server fails, user traffic can be delivered to the available server. A similar approach is to be used for accessing the Database Servers; there will be at minimum a pair of active Database Servers that synchronize state.

**Outages/Network Partitions:** In the case where the Client cannot access the Application Servers, a retry mechanism will be employed. This means that the Client will periodically attempt to connect to the Application Servers until a connection can be established. When this happens, a message informing the end-user should be displayed. Additionally, in the case where the application servers cannot access the Database Servers, a retry mechanism should be employed as well. In either case, application monitoring should be used to alert development and engineering staff.

1. **Security**:

**Secure Communication:** All communication between Clients and Servers will encrypted via strong SSL. Furthermore, any Personally Identifiable Information (PII) will be further encrypted using strong encryption while in transit to mitigate any Man In The Middle (MITM) attacks or Operational Security Issues.

**User Information:** User information will be protected across platforms by requiring the user to authenticate on each device. Using Java and the JVM’s cryptographic libraries along with SSL we can ensure that User PII is never transmitted in plain text. Authentication and authorization is provided via JWT and role based access controls. Further any PII data at-rest or stored in the database will encrypted and only decrypted on a need-to-know / least privilege approach.

**Operational Security**: Operational security will be provided by adhering to basic security principles:

* 1. Least Privilege
  2. Need-to-know authorization
  3. Physical security
  4. Operator background checks.
  5. Strong physical and network security.