

<Draw It or Lose It>

# **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 | 11/12/20 | Victor Feight | Executive summary, System architecture view |
| 1.2 | 11/13/20 | Victor Feight | Design constraints, domain model, evaluation |
| 1.3 | 11/14/20 | Victor Feight | Added recommendations |
| 1.5 | 11/29/20 | Victor Feight | Evaluate development needs and OS platform characteristics |
| **1.8** | **12/13/2020** | **Victor Feight** | Analyzed techniques and characteristics of system architectures, make a recommendation to The Gaming Room. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room has tasked us with porting their Android-only game, Draw It Or Lose It, to a web-game to serve Windows, Mac, Linux, and IOS platforms. To ensure a seamless cross-platform experience, we will outsource the programming portion to a team react-native developers to refactor the Android game into react-native for web. This will allow for a high quality, cross-platform interface on Android and IOS while making it possible to run React Native components and APIs on the web, using React DOM.

The game must have the ability to have one or more teams involved, with each team having multiple players assigned to it. Game names and team names must be unique so that a user may check if that name is in use when choosing a game, so we will use an Iterator pattern to ensure this. In addition, only one instance of the game may exist in memory at any given time, so we will use a Singleton pattern to ensure this.

## [Design Constraints](#_2et92p0)

*Software constraint*: The Android application is developed in Android JS using Javascript, HTML, and CSS. We intend have most of the code and logic refactored into React-native for web and this will increase development time.

*Software constraint:* Cross-browser compatibility. The intended application is to run on any system and browser, which means Android, IOS, Mozilla, Chrome, Firefox, and Internet Explorer. Cross browser testing is needed for compatibility issues that may arise.

*Software constraint*: User data management. The intended applications may hold certain user data privately and securely in a platform-independent way, and we must ensure secure coding practices such as using the “Draw It or Lose It API” for user login and authentication, with a RESTful approach.

*Hardware constraint*: A complete cross-platform program will require a team with familiarity not only in react-native-web, but in all modern browsers across Mac, Windows, Linux, Android, and IOS.

*Software constraint*: IOS and Android browser landing page. Since the program will be designed as an application available in the Google Play or Apple App store, each respective system will require a landing page with “Get it On the X Store” on Draw it Or Lose It’s mobile-interface homepage. The game should play when reaching the homepage on a Desktop browser, however.

## [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 ProgramDriver class holds the main function which creates a Singleton instance of the game in memory. It then uses another class, SingletonTester, to prove there is only one instance. The GameService class uses a public getInstance method to obtain the singleton private instance. This demonstrates both **encapsulation** which helps prevent instance data from being modified and hides implementation details, and it demonstrates the **Singleton** pattern which is used to create the one and only static instance of GameService.

The GameService class also demonstrates **polymorphism** through method overloading of the getGame function, which allows invocation of the correct method based on checking of method signatures. The addGame method of GameService demonstrates the **Iterator pattern**, which allows access to encapsulated Game instance without needing to know about the underlying list object data type.

GameService can have from 0 up to any amount of Game children classes. This demonstrates **composition**, since GameService *has-a* List<Game> type static variable. Likewise, Game demonstrates a has-aTeam relationship, and Team *has-a* Player. Every team and player must have a unique name, hence the addPlayer and addTeam methods shall use an **Iterator pattern**. The Entity class holds common attributes and behaviors from which Player, Team, and Game inherit; this demonstrates **Abstraction** and aids in code reuse.

****

## [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.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac includes built-in apache web-server commands a robust suite of command line tools such as python and commands to configure or access the server.  Highly intuitive Mac client creation policies compared to Windows. Cons: Runs on limited hardware configurations.  **Update:**  For the backend, if rolling your own server the project can start at minimal costs by running an HTML websocket that provides a client-side API to connect the browser with a **node.js** or **flask** backend server.  Note these costs are balanced among downtime costs if personal server crashes and security implementation, as an authentication method is needed to allow only authenticated Websocket sessions. CORs should be enabled, rate-limiting, payload size restriction, and using SSL over websockets chat and drawing in Canvas. | Linux also has a powerful command system and web-server, with the benefit of being much more cost-effective over Mac and Windows servers. Linux servers are stable and reliable, can run for long periods of time, and can be quite secure with good hardening practices.  **Update**:  Linux server best hardening practices include strong and unique passwords, multi-factor authentication with SSH keygen pair, updating server software regularly, close hidden ports, scan logs with Fail2Ban, disable unnecessary services, and utilize backups.  Digital Ocean, Vultr, Linode give a tiny and reliable Debian for cheaper than an AWS cloud solution that can be scaled more cheaply. Firebase can be used as a cloud-based database solution to scale to a larger number of users but expect higher costs. For example, Firebase can be used for user authentication with a Flask API, and we can use a multi-factor token login system using react web JSON objects. | Windows has a wide variety of software that can only be virtualized on other Oss. Users claim familiarity and ease of use, web extensions, and SQL support.  **Update**:  A windows server solution can also start at minimal costs rolling a personal Node.js or Flask server with RESTful authentication through React-native web based front-end. This will allow code portability across Browsers and Android, IOS systems This solution can start off with low cost and scale up if more computing power is needed Firebase or Amazon AWS/Heroku cloud based solutions.  Note Amazon offers **cloud-based** AWS microserver solutions, as does Heroku which will allow app to scale up infrastructure with more users. A heroku microservice could hold api functions to help **load balance** users amongst game rooms while ensuring minimal downtime with DDoS mitigation, spoofing protection, bad port scanning, and host-based firewalls. | The option to code the backend and host the mobile app is less viable than a Linux / Mac / Windows based solution. This option would not work well with the current specific skillset (react-native-web devs and Flask) we are seeking, as it would require a ground-up approach of coding and hosting the server In addition to cross-app development. |
| **Client Side** | Mac is generally costlier than Windows. Ease of use is about the same as Windows, requires a short to moderate amount of time to learn with an intuitive interface.  **Update:**  Consider hiring an independent React-native web developer for under $5k. A hybrid app can be created rather than a native app – allowing for portability (one code base, multiple platforms), faster speed to market and cheaper origination costs. | Linux is ideal for software and web developers because of its cost-effectiveness, and open source programs that work in unison with the system. However, maximum amount of time is required to learn compared to Mac and Windows defaults.  **Update**:  Since React-native web can can implemented to support iOS, Android, and Web platform with a single codebase, this solution is optimal for the front-end client interaction, and front-end JWT-based Authentication can connect to our Flask or Firebase database. The Canvas drawing and chat portion can be streamed over a secured Websockets living on our server or a Heroku instance. | Compared to Linux and Mac, Windows has many unique tools that can only be virtualized on the other systems, and has extensive support for web-app and and website development. Can also virtualize other OS.  **Update**: React-native for web codebase can also be worked on in a Windows environment using Visual Studio Code. Summary of development needs: **Flask dev, React native web dev, Security dev for proper authentication, upscale with cloud-based soluitions** | Provides clients flexibility of having the app anywhere, anytime. Requires adjustments in developing the app for screen real estate differences. All screen size possibilities for tablets, smartphones, and browsers should be accounted for. Mobile apps should have an intuitive interface designed for their small form factor.  **Update**: Utilizing react will allow responsive media adjustments to conform to IOS and Android phone form factors and scale up the interface for web. |
| **Development Tools** | VSCode for syntax highlighting and code previews, Homebrew package manager to install Unix and Mac utilities, Xcode IDE, iTerm2 emulator, Tower git client, Dash api browser. In addition, platform agnostic language set: Javascript, HTML, CSS, React, react-native, react-native-web. Chrome development tools, SQL.  **Update:**  **Flask, node.js, Websockets, mySQL** | Visual Studio Code, Atom, Vim, bash command line, git, node, flask.  In addition: Javascript, HTML, CSS, React, react-native, react-native-web, npm, yarn. Chrome development tools, MySQL  **Update:**  **Firebase, Amazon AWS, Heroku** | Visual Studio Code, Gvim, git for Windows, Git bash, node, npm, yarn. In addition: Javascript, HTML, CSS, React, react-native, react-native-web. Chrome development tools, mySQL | Browsers: Firefox, Opera, Samsung browser, Chrome, Metro browser. Website should work across all mobile browsers. Mobile browsers required for testing both the website and the app itself.  App: Javascript should be enabled on IOS and Android to allow app access. Google Play Store and Apple App Store. |

## 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**: The goal shall be to make Draw It Or Lose it completely cross-platform. The mobile front-end will be app-based, coded in React-native-web framework for the IOS and Android Apps. This will make it possible to run React Native components and APIs on the web, using React DOM for the browser front-end implementation. This will run on any browser in Linux, Mac, and Windows.

**Backend (server) Recommendation:** Mac x64-based server. Since the game is Android, Web, and IOS-based, compiling and testing code requires Apple’s SDK, and thus should be done from a Mac-based server, easing IOS development and allowing the use of Xcode for memory management. Android development requires Android SDK for mac, which is freely available.

1. **Operating Systems Architectures**: The server-side API language recommendation is Flask on a Mac x64 server, deployed in a virtual environment using virtualenv to ensure consistent dependencies. To ensure scalability and cost-effectiveness, a Heroku-based Function will manage load balancing, and images will be stored in an AWS S3 Bucket.
2. **Storage Management**: Include offsite automated, secure backup mirroring of server code and user data with compression; for dependency configurations, use Virtualized instances. Since network latency is a bottleneck for mobile and webapps, we can optimize performance through image optimization, code compression, and server-side caching for database queries.

**Recommendation for image server:** Amazon AWS Bucket, has all the above mentioned storage features, which will make scaling, backup, and security easier and more cost-effective long-term. They have pricing for infrequent – frequent access, allowing for highly scalable, fast, and relatively inexpensive data storage.

1. **Memory Management**: Debug React-native memory leaks using Android studio with Perf Monitor enabled. If MacOS is available, can use Xcode and enable “Leaks” under profile settings.

**Recommended technologies and approach**: Utilize web development and mobile application caching techniques, caching images as in-memory objects, serving images via Amazon’s AWS will allow faster downloads by reducing throttling done by browsers on the server. To reduce the requests browsers will make on subsequent visits, set a cache for credentials and certain assets for quicker load times. Optimizing images and utilize Local and Session Webstorage for persistent game data on a per-user basis. Prefetch assets using Manifests. Remove disconnected game sessions and objects from memory to avoid leaks.

1. **Distributed Systems and Networks**: To reach the most possible users, distribute the software from various sources: a web-page for the app will have a link to Facebook page and twitter page for development updates, Google Play Store download, Apple App store download, as well as Mac, Linux, and Windows packages in both x64 and x86 bit versions. Consider using Websockets for low-overhead communication between server and user (for ingame chat and image drawing). Use a Client-Server model with RESTful approach for user authentication. The server will maintain the shared game instances and images for each room, while the client will interact with the server through an API.

**Function As a Service**: With a high number of players, we may need several replicated servers, which sends connections to a Heroku-based Function As a Service to help load balance users across regions for matchmaking in game-rooms. We will have a database server to store user credentials, images can be stored in an Amazon AWS-bucket, and an event-based Flask server for the Drawing and chatroom logic. DigitalOcean spaces object storage is a cheaper, viable alternative with a bit more upkeep (requires Linux knowledge).

1. **Security**: Within the API, ensure data type validation, data format validation for JSON schema, data value validation. Encrypt access to critical user data on the database server. Ensure proper exception management is coded, handle any exceptions. Implement platform agnostic authentication with secure password recovery, using principle of least privilege for userbase. Ensure software libraries and dependencies are updated within virtualized instance, close unnecessary ports on the webserver. Ensure HTTP is implemented. Never give the client any controlling access to security information or database.

**Recommended technologies and approach:** Authentication of user will involve collecting user data in a RESTFUL approach (with JSON), preferably through a React or Angular javascript based front end that is portable across devices, and sending that data to our web server which contains the game logic. For example, the server may contain a Websocket on a Node.js to render images in a persistent way, since they must be drawn in real time. Messages for a chat application can also be built using a Websocket on a Flask server.