

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.0

## 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 | 10/23/2020 | Yuriy Kuptsov | Final version |

**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, owners of the Android app, Draw It or Lose It, is seeking our help in the development of the web-based version of their gaming app. The staff at The Gaming Room does not know how to set up the environment and they are seeing our assistance in streamlining the development of this web-based version. They have asked us to provide an initial software design document.

Immediately below is a summary of the client’s requirements:

* The game must support single and multiple team use.
* Each team must accommodate multiple players.
* The name of each instance of a game must be unique. The same holds for each team name. Users must be able to check whether a name is already in use when choosing a team name.
* Only one instance of a game is permitted to exist in memory at a given time. This may be accomplished by the use of unique identifiers for games, teams, or players.

Please see the section below on design constraints for more specific information about the web-based version.

## [Design Constraints](#_2et92p0)

* The project will require two different development environments. Rationale: Android and iOS have two different development kits, so that the project will require two different development environments.
* The development team will require all APIs for the current website; they will API will need to be translated to the different mobile environments (you implied this fact). Rationale: To promote a positive user experience, the development team will require all APIs for the current website. The development team will need to translate the API to the different mobile environments.
* The quality testing strategy for the project will be very complicated and time-consuming; this can affect the budget. Rationale: Apple controls both its software and its hardware, so the testing strategy for iOS devices will not be too complicated. Android, on the other hand, has more diverse mobile hardware which it supports. Therefore, the quality testing strategy for this platform will be very complicated and time-consuming. Setting up the quality environment alone could cause the project to go over budget.

## [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)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

**Summary**

The domain model consists of the following classes:

* ProgramDriver
* SingletonTester
* GameService – singleton class
* Entity
* Game – subclass of Entity, introducing polymorphism (see toString)
* Team – subclass of Entity, introducing polymorphism (see toString)
* Player – subclass of Entity, introducing polymorphism (see toString)

The point of entry for the program is within the ProgramDriver class. The ProgramDriver class uses the SingletonTest class to ensure that one and only one GameService instance is allocable.

The GameService class is a singleton class and implements the singleton pattern to ensure that only one stance of the GameService exists. The purpose of the GameService class is to allocate instance of Game such that each allocated instance has a unique name.

According to the client’s request*. “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.”* The requirement that each game’s name be unique, implies that multiple instances of the Game class coexist, but each with a unique name. Consequently, our interpretation is that “the game” refers to a uniquely named instance of Game and is not intended to imply that one and only one instance of the Game class may be instantiated at a time.

To implement the singleton class pattern, the GameService’s constructor is private. The constuctor is called only once at load time as the default value of a private instance variable. This variable’s value is accessible to the outside only by means of the public static method getInstance.

GameService introduces private static attributes games (a list of existing games), nextGameId (a counter used for generating IDs for instances of Game), nextPlayerId (a counter used for generating IDs for instances of Player), and nextTeamId (a counter used for generating IDs for instance of Team).

GameService provides additional public instance methods to add a game and to retrieve a game by name or id. When adding a game, GameService checks whether a game instance with the given name already exists within its list of games. If so, it returns it. If not, it creates a new game instance with the name and interns it into the list of games. GameService provides the following public methods: getInstance (mentioned above), addGame (retrieves a game with a given name if it exists and creates and interns (passing the result of incrementing the *nextGameId counter as its id)* one in its list of games otherwise and then returns the game instance), getGame with three overloads (one takes an index into the game’s list and returns the game at that index; another takes an id, searches games for an instance with that id, and returns it if it exists; and yet another that takes a name, searches games for an instance with that names, and returns it if it exists. The latter two overrides return null if the game they seek are not in the list of games), getNextPlayerId increments the nextPlayerId counter and returns it, and getNextTeamId increments the nextTeamId counter and returns it.

The SingtonTester is a simple class with one public method, singletonTester, that tests the singleton allocation of GameService and prints out the names of all existing Game instances.

The Entity class is a convenience class that provides the methods that are common to the Game, Team, and Player classes and introduces the private attributes required for the implementation of said methods. The private attributes it introduces are id and name. The public accessors it introduces are getId and getName. It also introduces the public toString method for use in representing the entities as strings (primarily used in printing to the console). The Entity class overloads its constructor. The default constructor is private and the public constructor takes an ID and name, setting the internal attributes accordingly.

Game extends Entity. Game introduces the instance variable teams, a list of existing teams. It introduces a private method getTeamIfExists in order to retrieve teams from the teams list (this method was added for convenience). Game’s constructor calls the constructor of its parent class, Entity, in order to allow Entity to perform any initialization of the instance needed by the Entity class. It also uses the GameService to get the next teamID. The Game class represents the internal state of a game and maintains its relationships with the instances of Team and Player involved in the game. Game introduces an instance variable, teams, which is a list of Team instances. Game also introduces the method addTeam. addTeam accepts a name of type String, looks up the name in its list of teams if it exists, otherwise it creates a new team with the given name and adds it to the teams list. In both cases, upon either retrieving an existing team with the given or creating a new team with the given name, it returns the team as its return value. Finally, the Game class overrides the toString method (introducing polymorphic behavior) to provide custom translation to a readable String format.

Team extends Entity. Team’s constructor calls the constructor of its parent class, Entity, in order to allow Entity to perform any initialization of the instance needed by the Entity class. The Team class represents the internal state of a team and maintains its relationship with the instances of Player involved in the game on behalf of the team. Team introduces an instance variable, players, which is a list of Player instances. Team also introduces the method addPlayer. addPlayer accepts a name of type String, looks up the name in its list of players if it exists, otherwise it creates a new player with the given name and adds it to the player’s list. In both cases, upon either retrieving an existing player with the given or creating a new player with the given name, it returns the player as its return value. Finally, the Team class overrides the toString method (introducing polymorphic behavior) to provide custom translation to a readable String format.

Player extends Entity. Player’s constructor calls the constructor of its parent class, Entity, in order to allow Entity to perform any initialization of the instance needed by the Entity class. The Player class represents the internal state of a player. Finally, the Player class overrides the toString method (introducing polymorphic behavior) to provide custom translation to a readable String format.

**Relationships Between Classes**

* ProgramDriver uses SingletonTester.
* GameServices has a zero to many relationship with Game.
* Game has a many to zero relationship with GameService and a zero to many relationship with Team.
* Team has a many to zero relationship with Game and a zero to many relationship with Player.
* Game, Team, and Player subclass Entity.

****

## [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** | Mac OS **LACKS** resource access control, subsystem isolation mechanism, integrated firewall, encrypted file systems, no execute (NX) page flag, and emulation. Mac has a low number of manufacturers acknowledge unpatched vulnerabilities. | Linux **HAS** resource access control, subsystem isolation mechanisms, integrated firewall (Netfilter), encrypted file system, no execute (NX) page flag, and emulation. Linux is open source and vulnerability patches emerge from the open source community unpredictably. | Windows **HAS** resource access control, subsystem isolation mechanisms, integrated firewall (Windows Firewall), encrypted file system, no execute (NX) page flag, and emulation. Windows has a low number of manufactures acknowledged unpatched vulnerabilities. Windows issues regular patches for security issues. | Most mobile devices lack the memory capacity and processor speed to host a web server. The web servers available for mobile devices are regarded as a novelty and not a practical solution to website hosting, especially for a high bandwidth distributed application with a large bandwidth footprint. Most mobile devices are connected to the internet via a wifi or 4g connection, neither of which are adequate or sufficiently reliable for the purposes of our client’s project. |
| **Client Side** | Since the client has stated that “The application must be delivered as a modern, responsive HTML interface running inside the web browser for desktop clients (Linux, Mac, Windows) and Linux, Max, and Windows all of adequate support for browsers capable of this requirement, the three are virtually equal in ability and cost per the client’s desire. This includes the salaries of client-side web developers.  However, anticipating that the professor wants an enumeration of other aspects of development on Apple, I have included following (see below).  The average Apple Software Engineer is paid $136,000 per year.  I am omitting the cost of graphic designers and content producers because they would likely not vary by platform.  Browser support for the Apple Platform is equal to Windows in terms of running web clients on the platform. | Since the client has stated that “The application must be delivered as a modern, responsive HTML interface running inside the web browser for desktop clients (Linux, Mac, Windows) and Linux, Max, and Windows all of adequate support for browsers capable of this requirement, the three are virtually equal in ability and cost per the client’s desire. This includes the salaries of client-side web developers.  However, anticipating that the professor wants an enumeration of other aspects of development on Linux, I have included following (see below).  The average Apple Software Engineer is paid $136,000 per year.  I am omitting the cost of graphic designers and content producers because they would likely not vary by platform.  Browser support for the Apple Platform is equal to Windows in terms of running web clients on the platform. | Since the client has stated that “The application must be delivered as a modern, responsive HTML interface running inside the web browser for desktop clients (Linux, Mac, Windows) and Linux, Max, and Windows all of adequate support for browsers capable of this requirement, the three are virtually equal in ability and cost per the client’s desire. This includes the salaries of client-side web developers.  However, anticipating that the professor wants an enumeration of other aspects of development on Windows, I have included following (see below).  The average Apple Software Engineer is paid $136,000 per year.  I am omitting the cost of graphic designers and content producers because they would likely not vary by platform.  Browser support for the Apple Platform is equal to Windows in terms of running web clients on the platform. | The average iOS developer earns $100,000 per year.  The average Android developer earns $96,000.  I am omitting the cost of graphic designers and content producers because they would likely not vary by platform.  The iOS operating system for iPhone, while versioned, is consistent across all versions and the hardware is consistent. This would reduce the cost of tailoring the client app to different models of the iPhone.  iPhone appeals to a wealthier demographic, in general. It has a smaller user base than Android, but that userbase may have more disposable income.  The Android operating system suffers from running on a wide range of hardware platforms. Unlike iPhone, the Android developer cannot assume that all of devices running their application will behave the same way on all hardware that runs Android. Android smartphone come in a wide range of memory, processor, and device configurations. Maintaining a client app on Android is a considerably more complex task than on Apple (iPhone). Moreover, testing an Android app requires running it on a wide range of devices.  Android has a much more diverse userbase than iPhone. Apps running on Android are likely to be more widely used and appeal to a larger market share. |
| **Development Tools** | Similar to the previous row, the development for tools for delivering a web-based client on Apple are similar to those used in Linux and Windows. It is unclear whether this question targets the developers on our team or The Gaming Room’s team; or whether it targets the production of the aforesaid web applications regardless of what platform is used to develop them.  In terms of a recommendation on what platform should be used to develop the web based client’s, Apple is a good candidate because it supports a wide-range of content creating tools and IDEs for web client development.  Most artistic content creators prefer Apple and having them work together with developers on the same kind of platform can make the interoperability of the teams more seamless.  As for the developers, they will be writing HTML, CSS, JavaScript (or one of its derivatives such as TypeScript), and may be using a framework such as Backbone, Angular, or any of the many others that have emerged.  Apple supports a wide range of IntelliJ based IDEs for HTML, CSS, and JavaScript. The skill set involved is not specific to Apple.  Example IDE’s for developing HTML based web-clients using Apple as a development platform are: Visual Studio Code (yes, it runs on Apple), Komodo IDE, and Netbeans.  Visual Studio Code is available in a free edition. Netbeans is free. Kobodo IDE has a licensing cost. | Similar to the previous row, the development for tools for delivering a web-based client on Linux are similar to those used in Apple and Windows. It is unclear whether this question targets the developers on our team or The Gaming Room’s team; or whether it targets the production of the aforesaid web applications regardless of what platform is used to develop them.  In terms of a recommendation on what platform should be used to develop the web based client’s, Linux is an inferior platform because most of the content creation tools available on Linux are inferior to those found on Mac and Apple. For example, instead of using Photoshop, Adobe Illustrator, or Adobe Animate, one would be forced to rely upon GIMP and Blender. Both of which have a long learning process due to the fact that they were created by random developers all of which have their own idea of how each tool’s UIX should appear, causing confusion and a lot of learning to use each tool within the application.  On the plus side, GIMP and Blender are free, but the employer would likely end up paying more in terms of salaries as the work would take longer to complete.  Of course, artistic designers could use Apple to produce these artifacts, but if they will be using Apple, then developers may need to deal with cross-platform issues.  As for the developers, they will be writing HTML, CSS, JavaScript (or one of its derivatives such as TypeScript), and may be using a framework such as Backbone, Angular, or any of the many others that have emerged.  Linux supports a wide range of IntelliJ based IDEs for HTML, CSS, and JavaScript. The skill set involved is not specific to Apple.  Example IDE’s for developing HTML based web-clients using Apple as a development platform are: Visual Studio Code (yes, it runs on Apple), Komodo IDE, and Netbeans.  Visual Studio Code is available in a free edition. Netbeans is free. Kobodo IDE has a licensing cost. | Similar to the previous row, the development for tools for delivering a web-based client on Windows are similar to those used in Apple and Linux. It is unclear whether this question targets the developers on our team or The Gaming Room’s team; or whether it targets the production of the aforesaid web applications regardless of what platform is used to develop them.  In terms of a recommendation on what platform should be used to develop the web based client’s, Windows is a good candidate because it supports a wide-range of content creating tools and IDEs for web client development.  Most artistic content creators prefer Apple and having them work together with developers working on the Windows platform may introduce some small issues, but these issues have been tackled by literally thousands of other enterprises and their solutions are readily available.  As for the developers, they will be writing HTML, CSS, JavaScript (or one of its derivatives such as TypeScript), and may be using a framework such as Backbone, Angular, or any of the many others that have emerged.  Apple supports a wide range of IntelliJ based IDEs for HTML, CSS, and JavaScript. The skill set involved is not specific to Apple.  Example IDE’s for developing HTML based web-clients using Apple as a development platform are: Visual Studio Code (yes, it runs on Apple), Komodo IDE, and Netbeans.  Visual Studio Code is available in a free edition. Netbeans is free. Kobodo IDE has a licensing cost. | Most mobile device development is performed on a desktop computer and then tested on a mobile device. Apple provides good tools for iPhone development on their desktop. Microsoft Visual Studio provides good tools for both Android and iPhone development on the Windows platform. Free tools are also available on Linux for development and deployment of iPhone and Android apps.  Apple provides the Xcode IDE for developing apps for Mac, iPhone, iPad, Apple Watch, and Apple TV. xCode is free.  Microsoft provides Visual Studio for developing mobile apps deployable to iOS, iPhone, Android, and Windows. Visual Studio allows integration with Unity for animation, and has an emulator for Android. Visual Studio runs on Apple and Windows. It does have a licensing fee.  Visual Studio, since it runs on Apple and Windows, presents no additional problems for the interoperability of the design, artistic, and development teams. |

## 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: Recommend an appropriate operating (server) platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments.**

Our recommended platform for your application is Microsoft Azure using a Windows Server, ASP.NET, Visual C#, and Microsoft SQL Server. Microsoft Azure would be a good choice for many reasons[[1]](#footnote-1).

* Moving on from your current environment to a distributed network environment would be simplified by taking advantage of Microsoft Azure’s cloud computing service. The Azure cloud computing service removes the need of hosting your own servers. This will save you money on hardware, hardware maintenance, and the need for staff trained in hardware and server maintenance.
* Microsoft Azure offers a wide range of virtual servers, infrastructure, and many levels of security – all built into their platform. With their reputation on the line and with their expertise in security, your company can benefit from Azure’s sophistication in security without needing to bring in your own security experts.
* As your user base grows, expansion of server capacity becomes the responsibility of Microsoft Azure, avoiding any sudden need for additional servers, bandwidth capacity, and additional staff to maintain a growing number of servers.
* ASP.NET is well supporter by Microsoft. It provides many tools and services for delivering dynamically created web-pages. Visual C# is a powerful Java-like language that can be integrated easily with ASP.NET and Microsoft SQL Server. The .NET environment allows for interoperability between many different programming languages. Finding expertise in ASP.NET, Microsoft SQL Server, and Visual C# is not difficult.

**Operating Systems Architectures: Describe the details of the chosen operating platform architectures.**

Your servers will be under the management of Microsoft Azure. They will be virtual servers and their security and maintenance will be the responsibility of Microsoft. ASP .NET is a server-side web-application-framework. It is designed for web development. It allows you to produce dynamic web pages and is well suited for web-based applications.

In the next section, we will discuss our recommendations for storage management. However, the primary link between our application and our storage mechanisms will be Microsoft SQL server. Working with both ASP .NET and Microsoft SQL server, Visual C# will be our development environment for the gaming model and for the viewer model integrated with ASP.NET. Our model, in Visual C#, will communicate with Microsoft SQL server to obtain and store data and images. It will also communicate with ASP .NET to provide bindings to the data through Visual C#, which will translate the models into views that inform ASP .NET’s HTML5 generating methods.

**Storage Management: Identify an appropriate storage management system to be used with the recommended operating platform.**

Microsoft Azure is a cloud-based computing environment. Our images and data will be stored in encrypted form on Azure’s cloud. More details about security can be found in the final section, however, keep in mind that Microsoft Azure’s cloud is built with high security in mind. Storing our images and data on the cloud will liberate your company from needing to purchase, maintain, backup, and possibly expand its storage capabilities.

**Memory Management: Explain how the recommended operating platform uses memory management techniques for the Draw It or Lose It software.**

As mentioned in the previous section, all of our data will reside on one Microsoft SQL Server. Microsoft Azure provides load-balancing across multiple servers. Your game will run on at least one Microsoft virtual server, but if the load increases, a copy of your game will be duplicated on as many additional virtual servers as needed. All instance of your game will communicate with the same Microsoft SQL Server and the game application will be designed to synchronize data requests and storage into that one database.

The key reason for choosing Visual C# as a layer between ASP .NET and Microsoft SQL server, is to allow a sophisticate level of logic between the serving up of webpages and the data stored in the database. This layer, in addition to representing the model behind the game and producing viewer objects which are translated into APS .NET, will provide the locking mechanisms to avoid conflict between the data requests and updates to the database server. It will also use a caching mechanism to keep the most frequently used data records in the application’s memory, acting as a kind of objected oriented database with an SQL backend. Those records and images that are most frequently requested, will be held in the application’s memory and only request of the database when or if the data on the database changes or they have fallen into disuse for a period of time while other data records have become more frequently accessed. The object-oriented database layer, written in C#, standing between ASP and the database, will always maintain and manage a cache of the most frequently requested assets.

Finally, to minimize the overhead of requests for assets upon the server, an AKAMAI service will be used to cache assets on the web and will only request those assets again when they have changed or fallen out of frequent use.

**Distributed Systems and Networks: Knowing that the client would like Draw It or Lose It to communicate between various platforms, explain how this may be accomplished with distributed software and the network that connects the devices. Consider the dependencies between the components within the distributed systems and networks (connectivity, outages, and so on).**

By using Microsoft Azure, we have placed much of this issue into Microsoft’s hands. Azure will be responsible for maintaining connectivity and handling outages. As for how the communication will be accomplished, let us consider the different modes of communication employed by the game. Requests for pages will be made through the internet in the form of HTTPS get and put requests. Any messaging between players will utilize Microsoft Azure’s Service Bus which provides messaging queues. Players will be able to communication through this channel directly, without passing through our servers.

**Security: Security is a must-have for the client. Explain how to protect user information on and between various platforms. Consider the user protection and security capabilities of the recommended operating platform.**

By using Microsoft Azure and taking advantage of its multilayer security approach, your company will benefit from the highest level of professional security without the need to hire your own team tasked to that responsibility. Microsoft Azure provides the Azure Application Gateway, a web application firewall, to protect against common attacks (e.g. SQL injection, cross-site scripting attacks, and session hijacking). Also provided is App Service Authentication / Authorization. This feature provides role-based authentication of both game users and your own staff (i.e. administrators, etc). Storage is also protected through role-based access control and encryption of stored data. Messages between The Gaming Room and its users and between users themselves are protected by encryption in transit (e.g. HTTPS and SMB 3.0 encryption). We can also offer you two-stage authentication. Finally, Azure provides the Azure Security Center to enforce and monitor security issues and policies relating to your application.[[2]](#footnote-2)

1. Microsoft Azure: https://en.wikipedia.org/wiki/Microsoft\_Azure [↑](#footnote-ref-1)
2. <https://docs.microsoft.com/en-us/azure/security/fundamentals/services-technologies> [↑](#footnote-ref-2)