Faculty of Engineering and Applied Science

SOFE 3650U Software Design and Architecture CRN 44212

Group 20 Final Phase for Video Game Website

Name	Student #
Rubbia Pasha	100702075
Alexander Campbell	100703650
Atharshan Kennedy	100590243
Joey Villafuerte	100759003

ADD Step 1: Review Inputs:

Design Purpose

This architecture is for a scalable, robust online game hosting website. The purpose of this design is to thoughtfully recreate a past website except make it more robust, easier to track data, and add various features that would increase user engagement and retention.

Primary Functional Requirements

These are the primary functional requirements, meaning they are the highest priority use cases.

Use Case	Reasoning
UC-1	Directly supports core business through monitoring various activities that are needed for users to be able to use the website as intended
UC-2	Directly supports core business so that it can detect faults and allow for precaution measures to be taken into effect
UC-5	Directly supports core business so that it can solve configuration and the technical issues associated
UC-7, UC-8	Directly supports core business so that data-driven improvements and fixes can be made

Quality Attribute Scenarios

QA-1, QA-2, QA-3 and QA-5 are selected as drivers

Scenario ID	Importance to Customer	Difficulty of Implementation
QA-1	HIGH	HIGH
QA-2	HIGH	HIGH
QA-3	HIGH	HIGH
QA-4	MEDIUM	MEDIUM
QA-5	HIGH	MEDIUM
QA-6	LOW	MEDIUM
QA-7	MEDIUM	LOW

Constraints

CON-3, CON-8, CON-5, CON-6, CON-7 and CON-12 are selected as drivers

Architectural Concerns

All concerns listed below are selected as drivers

ID	Concern
CRN-1	Establishing an initial website architecture that supports backwards compatibility with existing games and existing users.
CRN-2	Leverage Django to make the back-end of the website in a time efficient manner, and React.js in the front-end in order to make the user experience more interactive.

Iteration 1: Establishing Overall Website Architecture

The first iteration outlines the initial results of the design process for the ADD steps. In the first iteration, we are thinking very high-level. We are simply choosing what would be the best general reference architecture based on the drivers. We considered several, but ultimately decided that the **Rich Internet Application** was proven to be the most useful reference architecture for this application.

Step 2: Establish Iteration Goal by Selecting Drivers

Driver selection was based on what influenced the structure of the system. This helps establish the iteration goal of achieving the architectural concern of establishing an overall system structure. Drivers listed below:

Driver Table

Driver ID	Driver Content
QA-1	Security
QA-2	Availability
QA-3	Performance
QA-5	Modifiability
CON-3	The user must always be able to reliably play games that have been loaded, even when the connection is very poor
CON-8	When storing a user's info, it must be secret/encrypted and only the user that the data is referencing should have access to it.
CON-5	The website must be accessible by several web browsers (Chrome, FireFox, IE) and support use from multiple operating systems (Windows, Mac OS, Linux).
CON-6	Users must be able to open a game of their choice within three pages from any point within the website.

CON-7	The website will be up to date and running at all times for both low and high influx of players.
CON-12	Before a user can make changes to their account that are pushed to the server, the user must confirm the changes via a single pop-up window.

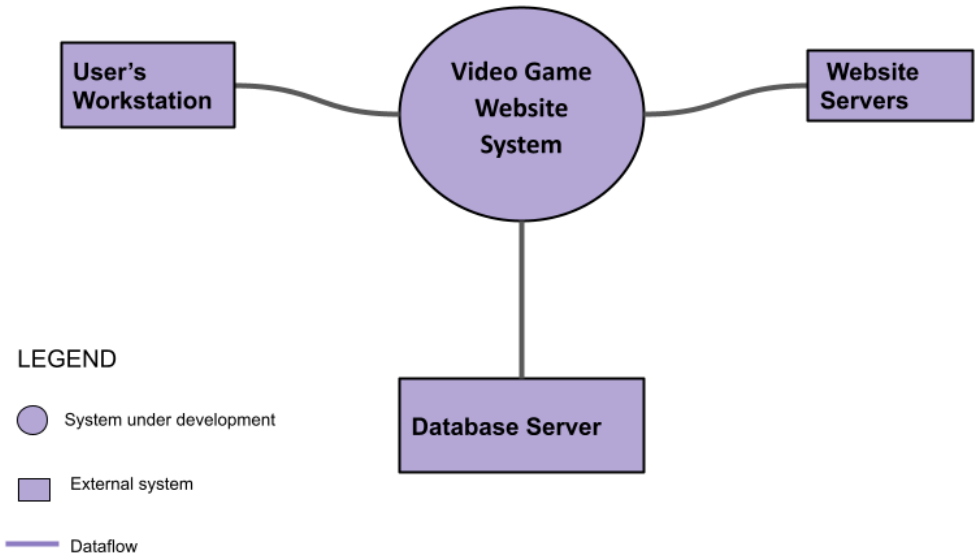

FIGURE 1: Context Diagram for the Video Game Website System

Step 3: Choose One or More Elements of the System to Refine

This is a brownfield development effort, so in this case the element to refine is the entire video game website system, which is shown in Figure 1. In this case, refinement is performed through decomposition.

Step 4: Choose One or More Design Concepts That Satisfy the Selected Drivers

The following table summarizes the selection of design decisions

Design Decisions and Location	Rationale
Logically Structure the client and server part of the system using the Rich Client Application reference architecture.	The Rich Internet Application (RIA) supports applications that typically run inside the browser of a user and tend to be a bit more complex than a generic web application. RIA's are very useful for web applications that need a rich user interface, which most video games require. They also allow for some processing to be done on the client-side,

meaning that we can efficiently use both the client and server in order to create the smoothest user-experience possible (CON-3). This will also support QA-3 such that we can allocate certain resources to the server or client as needed. Having a rich user interface also allows us to track how a user interacts with the website better (UC-1).

Discarded alternatives:

Alternative	**Reason for Discarding**
Web Applications	This reference architecture is mainly used for applications where most of the application resides on the server-side and a rich user interface is not needed. For a video game hosting website, client processing capabilities and the ability to support a rich user interface for various games is necessary.
Rich Client Application	Though this reference architecture supports a responsive and interactive user interface and intermittent network connection, which would support several drivers well, it does not have any web connectivity.
Service Applications	This reference architecture does

		not provide a user interface but rather expose services that are consumed by other applications. Therefore the use of this application is used by humans so the need of a user interface would be necessary. Plus it would cause more complications as well.
	Mobile application	This reference architecture is oriented toward the development of applications that are handheld devices. This alternative was discarded because there are many limitations that cause this option to not be viable nor practical, such as the screen size, and the resources that would be available.

Framework Decisions:

Design Decision and Location	**Rationale**
Build out the backend using Django to serve the web pages that are built using ReactJS	Django and React are both very widely used frameworks, which allows the dev team to build out the website quickly. Django also has many built-in security protections (QA-1)
Use Heroku to deploy the website	By deploying the website through a provider, we can get access to their dynamic allocation depending on the amount of users we have (CON-7). Heroku also has various built-in tools for measuring website performance that can be easily implemented (UC-7, UC-8).

Step 5: Instantiated Architectural Elements

For the most part, the RIA reference architecture suits our use well. The only difference is that we will be making use of the Isolated Storage.

Design Decision and Location	**Rationale**
When possible, store data locally in the client	The website must be playable even when connection is poor. We can save data that is needed to continue the game locally even when internet connectivity is poor (CON-3). Following the reference architecture, we can adapt the optional Isolated Storage as a temporary data store while internet connectivity is poor.

Step 6: Sketch Views and Record Design Decisions

The diagram in Figure 2 shows the sketch of the module view of the two reference architectures that were selected for the client and server applications. These have now been adapted according to the design decisions we have made.

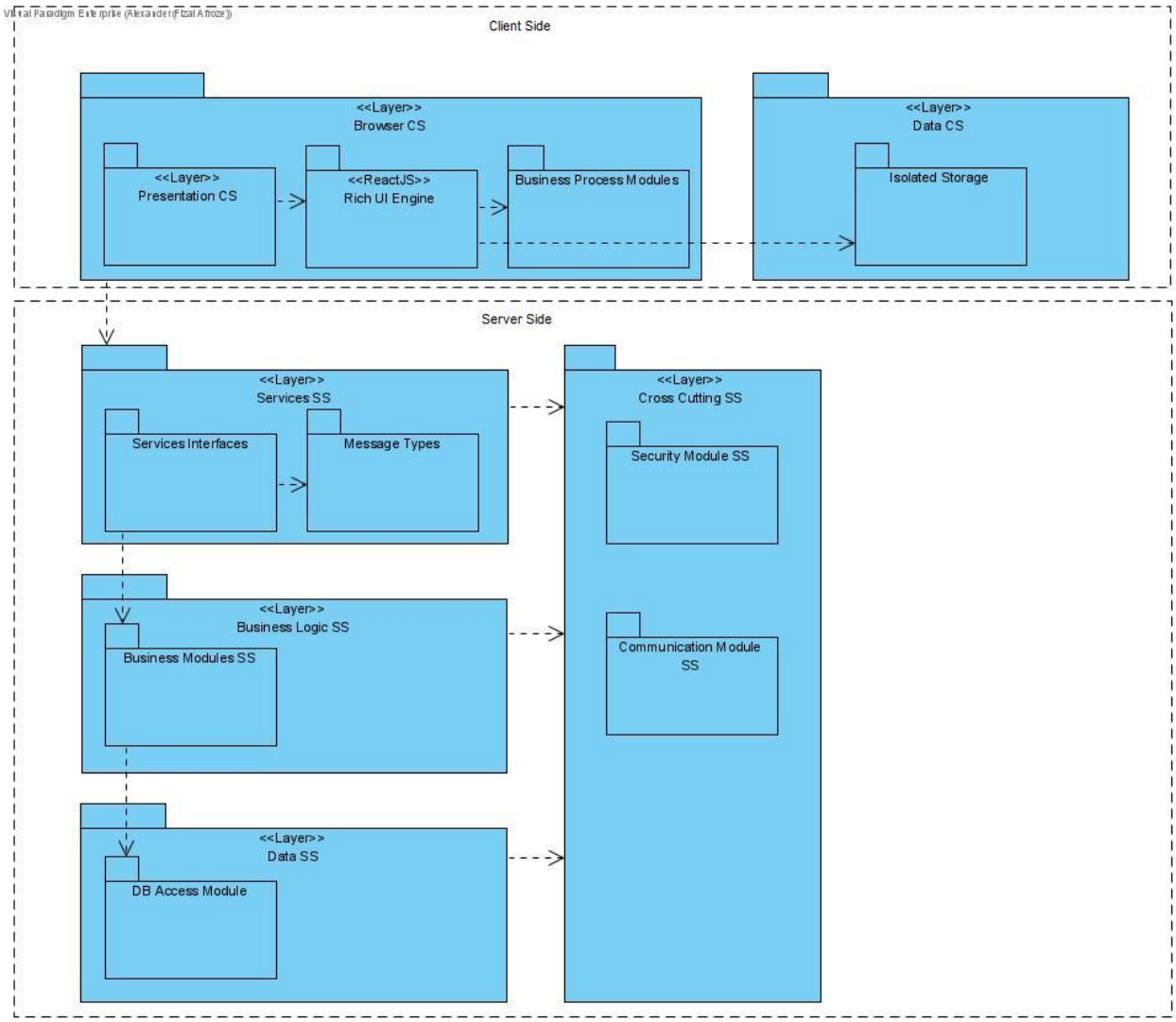

FIGURE 2: Sketch of a module view of client side and server side architecture

Each of the elements that have been selected is explained with a short description of its responsibilities. The following table summarizes the information that is captured in Figure 2:

Element	**Responsibility**
Browser CS	This layer contains modules for how the user interacts and manipulates the website
Presentation CS	This layer contains modules that control how the user can interact with the UI
Rich UI Engine	Contains modules that are responsible for more sophisticated UI interactions
Business Process Modules	This module contains modules that can perform business

	processes that can be executed on the client
Data CS	This layer is responsible for holding data on the client side
Isolated Storage	These modules are responsible for containing data that allows for users with intermittent internet to continue playing
Services SS	This layer contains modules that can expose services for the client use
Services Interface	This module contains services that are exposed for client use
Message Types	This module contains services that are specific to managing the types of messages that a client and server exchange
Business Logic SS	This layer contains modules responsible to implement business operations and other business logic
Business Modules	This module is responsible for implementing business operations and other business logic
Data SS	This layer contains modules responsible for data storage and retrieval for other layers
DB Access Module	This module is responsible for communication between the Server and the Database
Cross Cutting SS	This layer contains modules that range in functionality from security to various communications that are needed between layers
Security Module	This module is needed for implementing various security measures such as encryption, logging, etc.
Communication Module	This module is responsible for various communications between various layers

Figure 3 shows the initial deployment diagram and where elements from figure 2 are supposed to be within the deployment diagram. Also, the table below lists the responsibility of each component for the initial deployment diagram in figure 3.

Element	**Responsibility**
User workstation	The users devices that can host the client side logic of the system.
Application Server	The server that can host the server side logic of the web application.

Database Server	The server that contains and hosts the relational database of the system

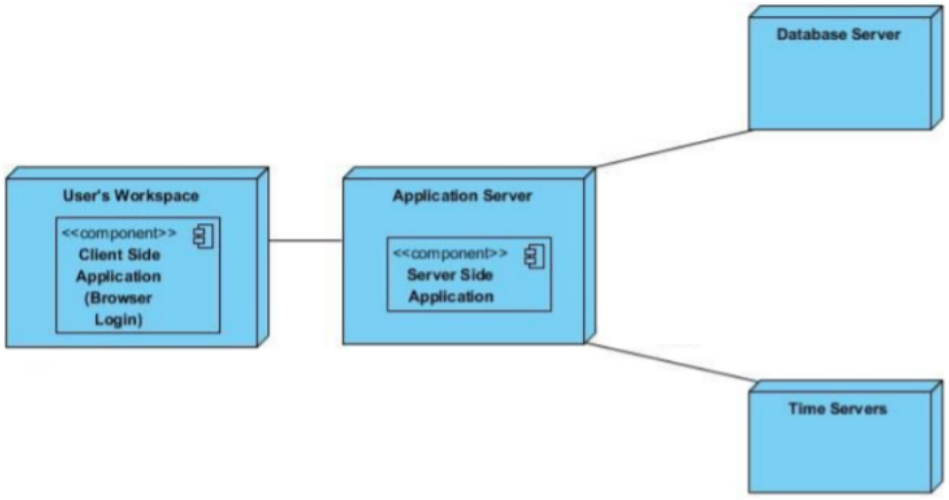

FIGURE 3: Deployment diagram for the video game website system

Step 7: Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

Not Addressed	Partially Addressed	Completely Addressed	Design Decisions Made during Iteration
	QA-1		The security module implemented in the selected reference architecture that supports this functionality
QA-2			No relevant decisions made
QA-3			No relevant decisions made
QA-5			No relevant decisions made
	CON-3		The Data SS layer is implemented in the selected reference architecture that supports this functionality. More information about how the data is saved must be decided

	CON-8		The security module implemented in the selected reference architecture that supports this functionality.
		CON-5	Using Django along with ReactJS should allow for this website to run on all popular browsers
CON-7			No relevant decisions made
CON-12			No relevant decisions made
		CRN-1	Using Django will allow for the older web pages to be served alongside the newer web pages with minimal new interfacing needed. In addition to this the reference architecture was selected.
		CRN-2	Django and React have been selected and should encapsulate all uses within the Package Diagram

Iteration 2: Identifying Structures to Support Primary Functionality

This section presents the results of the activities that are performed in each of the steps of the ADD in the second iteration of the video game website. Here, the goal is to identify each of the support structures needed for the primary functionality.

Step 2: Establish Iteration Goal by Selecting Drivers

For this iteration we will be considering the primary use cases as they best describe the primary functionality of the website. The use cases in mind are:

- UC-1: Monitor Activity
- UC-2: Detect Fault
- UC-5: Configuration
- UC-7: Collect Website Data
- UC-8: Analyzing Data

Step 3: Choose One or More Elements of the System to Refine

The elements that will be refined in this iteration are modules located in the different layers that are outlined by the reference architectures from the previous Iteration, such as the Rich Client Application reference architecture.

Step 4: Choose One or More Design Concepts that Satisfy the Selected Drivers

Design Decisions	Assumptions and Rationale
Create a Domain Model for application	We choose to create a Domain Model for this application to define each of the major domain objects that reflect our functionality. This is done to ensure that we can create an efficient architecture from the very start and find any issues or redundancies early on.
Identity Domain Objects that encapsulate our primary functionality	Domain Objects must be created to ensure that all primary functionalities of our application are considered in this process. By mapping our primary functionalities to specified Domain Objects, we can easily consider if all of our required functions have been planned.
Divide the Domain Objects into specialized modules	Instead of having large, encompassing Domain Objects, it's much easier to design and develop smaller, specialized modules that have specific use.
Use Django with no external ORM framework	Django is a popular, "batteries-included" Python framework mostly used for back-end web development. Due to its "batteries-included" nature it supports database queries through its **models** library (UC-1, UC-2, UC-5). Django can also be used to collect data and create custom data visualization suites, supporting UC-7 and UC-8. A discarded alternative was MongoDB. The team decided against its use as Django's models library was powerful enough and it was decided to keep the application as lightweight as possible for easier maintenance.

Step 5: Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

The instantiation design decisions made in this iteration are summarized in the following table:

Design Decisions	Assumptions and Rationale
Create an minimum viable domain model	Only primary use cases should be considered for an initial creation to avoid excess work. If we add superfluous modules that are removed in the end, it is only wasted work.
Decompose the domain objects across the layers to identify layer-specific modules with an explicit interface	This technique ensures that modules that support all the functionalities are identified. This arises an architectural concern CRN-3: A majority of modules shall be unit tested. Only a majority of modules are being covered by this concern because the modules that implement user interface functionality are difficult to test independently.

Step 6: Sketch Views and Record Design Decisions

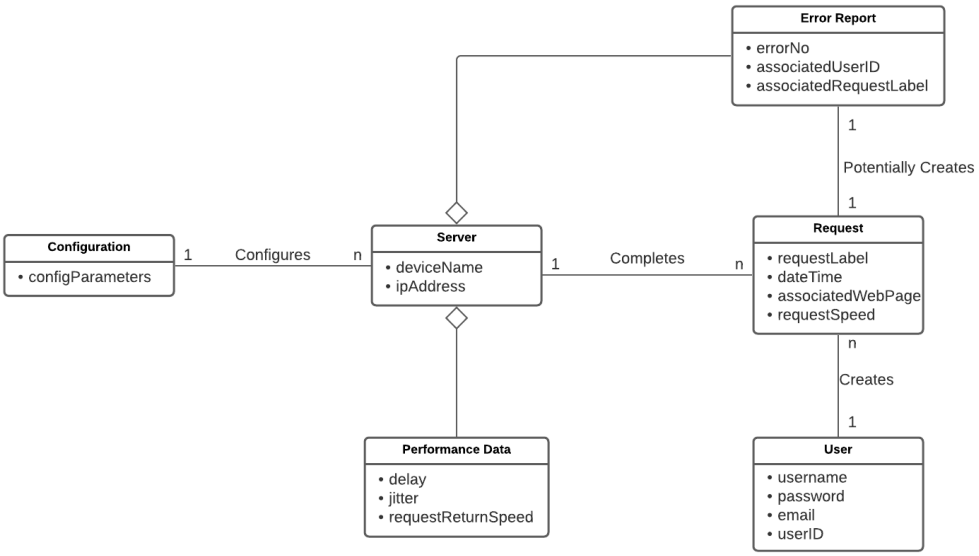

Figure 4: shows an initial domain model for the system

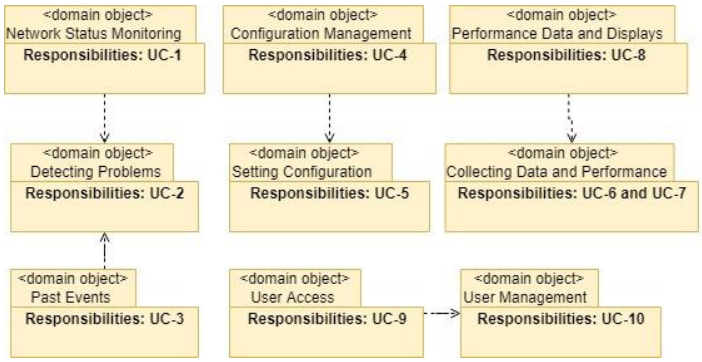

Figure 5: Domain objects associated with the use case model

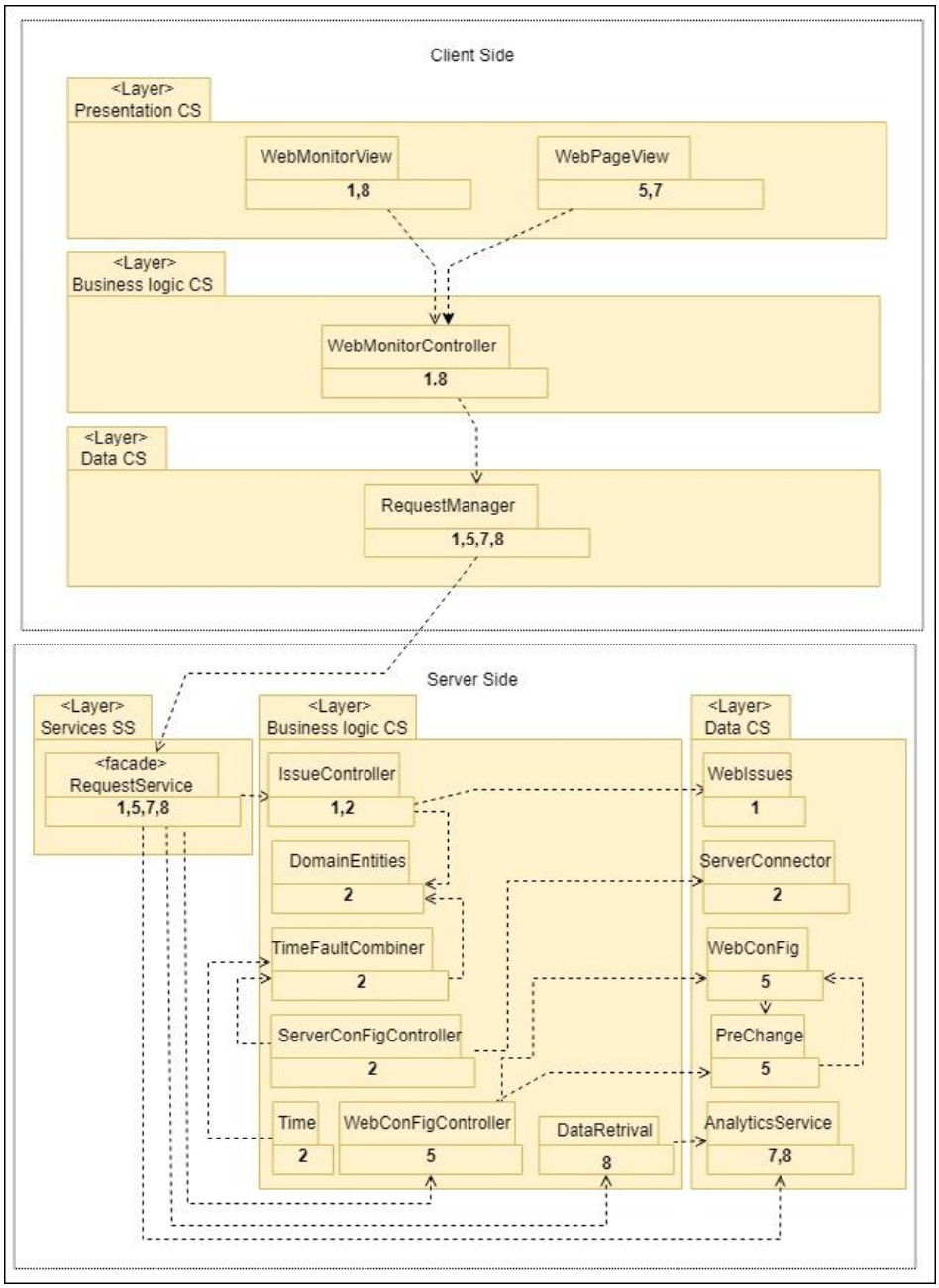

Figure 6: Primary Uses Case supported Modules

The following table on the next page describes the responsibilities of each element in the above Primary Uses Case supported Modules diagram.

Element	Responsibility
WebMonitorView	Displays an interface for IT and allows for new view updates to be displayed after a request has been processed.
WebMonitorController	Allows for information to be provided for the presentation layer that represents the current status of the website.
WebPageView	Displays an interface for the Player's and Admin which allows for interaction with the website.
RequestManager	Takes in user requests and communicates with the server side.
RequestService	Receive requests from clients and acts as a facade for different types of requests.
IssueController	Provides the logic that is necessary for requesting website issues.
DomainEntities	Houses the entities that reside within the server side.
TimeFaultCombiner	To create a new data entry that has the current fault and time.
ServerConFigController	Provides the business logic needed for listening to events.
Time	Help aid with tracking time when faults occur.
WebConFigController	Provides the logic that is needed for updating the web configuration and returning back the result to the request service.
DataRetrival	Uses the logic that is needed for retrieving performance data and sending back to request service.
WebIssues	Collects issues/faults that reside with the website itself or from Player's request that gets sent to the server.
ServerConnector	Allow for communication to be sent when the server detects an event or fault has happened.
WebConFig	Holds the current website configuration that is global to all users.
PreChange	Collectes the previous state of the website first before applying any updates in order to prevent breaking the site.
AnalyticsService	Collect data needed for performance that can then be extracted and viewed by an IT.

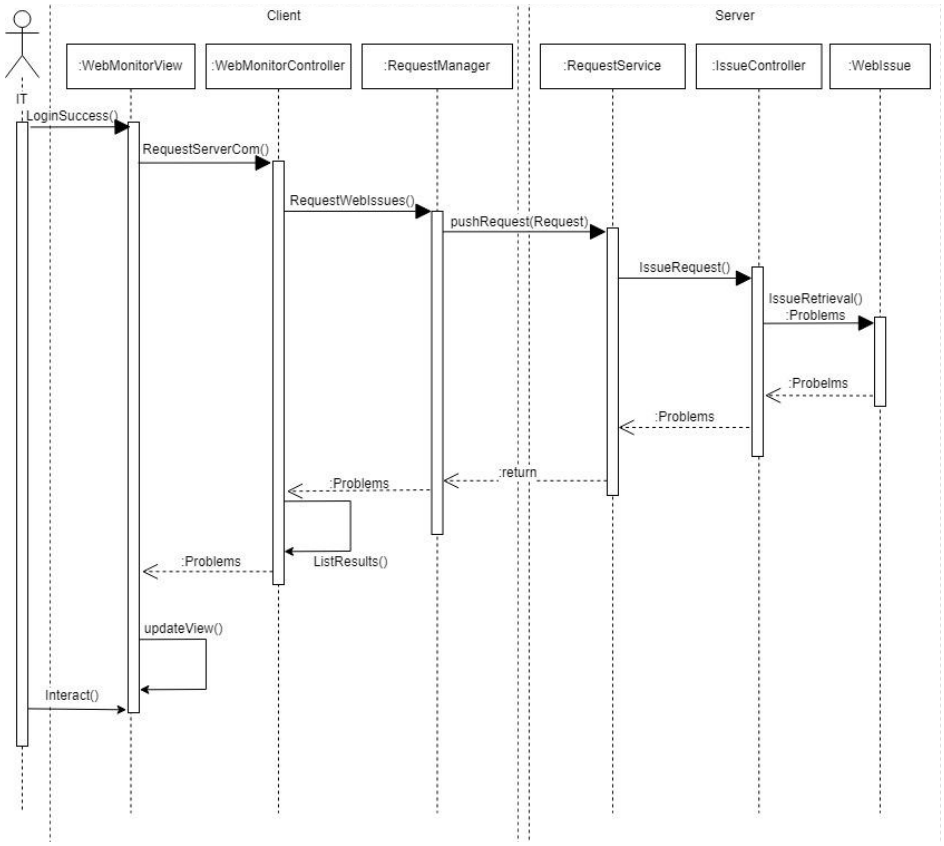

Figure 7: Sequence diagram for UC-1

The following table defines the interacting elements of the above Sequence Diagram for UC-1.

Element	**Method**	**Description**
WebMonitorView	LoginSuccess() interact() updateView()	Upon Login success IT is provided with interactive lists of issues.
WebMonitor--Controller	Problems RequestServerCom() ListResults()	Provide the user with website issues for the user to interact with after listifying the results.
RequestManager	Problems RequestWebIssues()	A request for web issues is made which returns the problems of the website.
RequestService	Return pushRequest(Request req)	A simplified interface that retrieves a request and is the only method that interacts with the ClientRequestReciver.
IssueConroller	Problems IssuesRequest()	Current pending issues of the website, upon request, are returned back. This allows for a complete look through into any problems related to the website by the user.
WebIssue	Problems IssuesRetrieval()	Returns a group by list of pending issues.

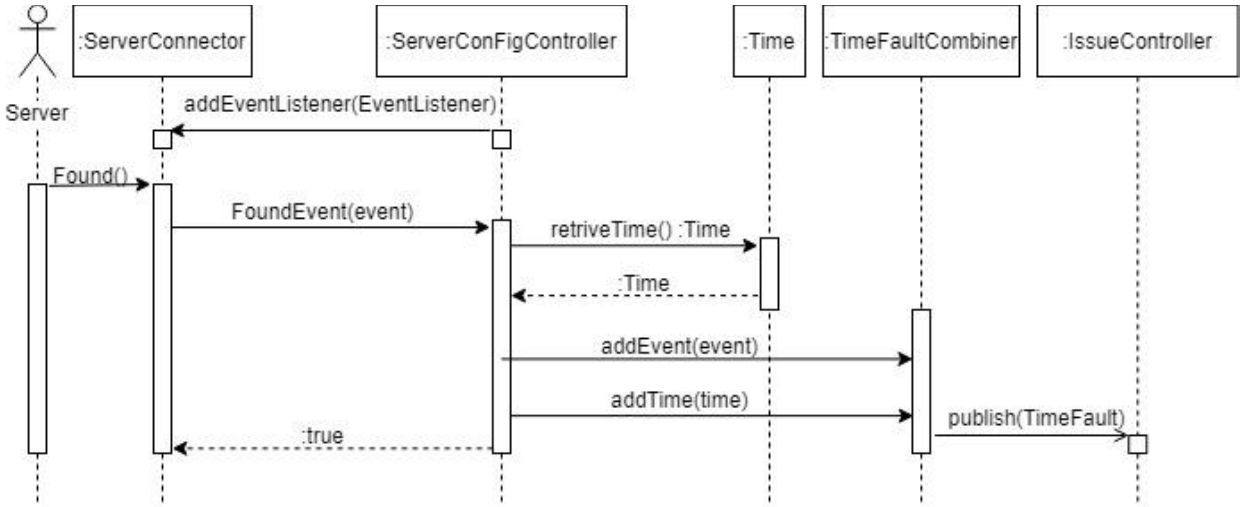

Figure 8: Sequence diagram for UC-2

The following table defines the interacting elements of the above Sequence Diagram for UC-2.

Element	**Method**	**Description**
ServerConnector	Boolean addEventListener(EventListener e)	Components within the business logic are registered as listers for the events that come from the server.
ServerConfigConnector	Boolean FoundEvent(Event et)	Upon an event being received this callback method will be invoked. True is sent back to allow for a new fault to be detected.
TimeFaultCombiner	addEvent(Event et) Time addTime(time t)	Takes the time and fault event and sends it as a fault with a fault time of when it happened to the system.
IssueController	publish(TimeFault tf)	Notifies the system of the detection of a fault.

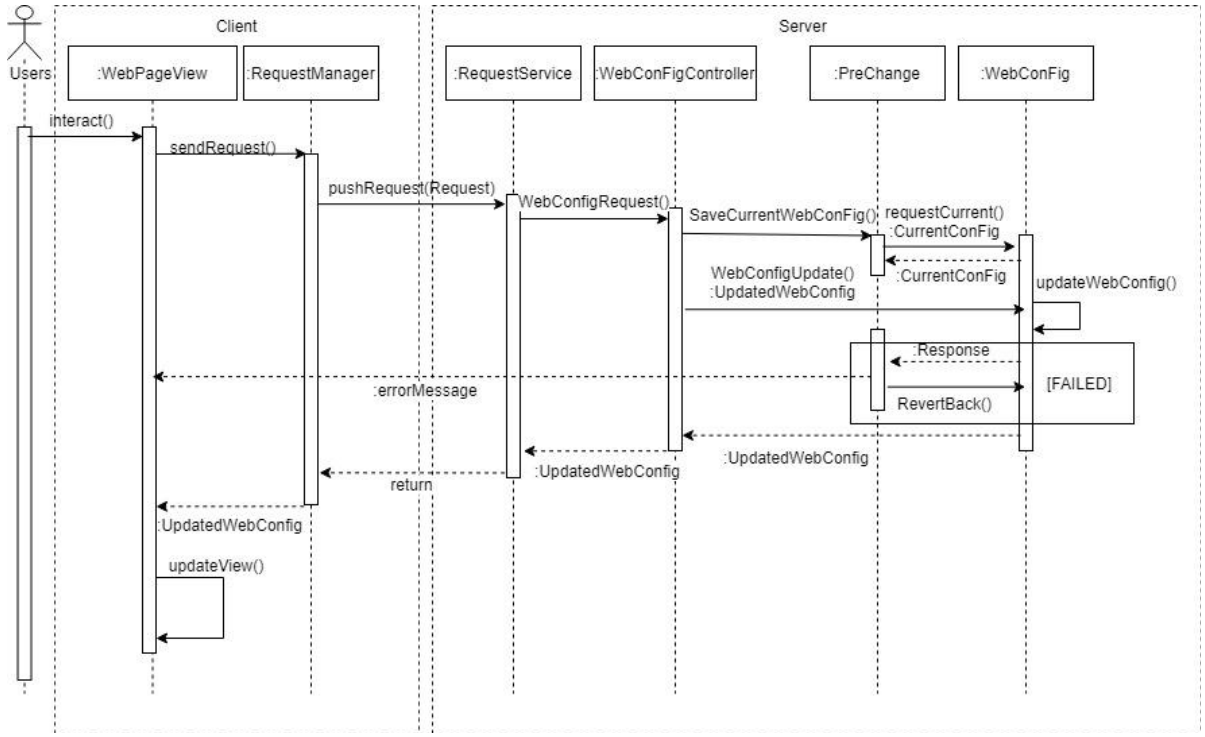

Figure 9: Sequence diagram for UC-5

The following table defines the interacting elements of the above Sequence Diagram for UC-5.

Element	**Method**	**Description**
WebPageView	interact() updateView()	The import of a new game or web config change request is done through WebPageView. The user can interact with a passed or failed view after update.
RequestManager	UpdatedWebConfig senRequest()	When a request is made for configuration change of website the user will have a new or old configuration of website benign sent to them.
RequestService	Return pushRequest(Request)	The config request is pushed to the server side. Once a process has been made the result is returned to Client.
WebConFig--Controller	UpdatedWebConfig WebConfigRequest()	Performs the operations needed to safely change the web config upon a request from the user. The updated web config is sent back to requestService.
PreChange	errorMessage SaveCurrentWebConFig()	The current web config is asked for backup before changes are applied. A failed message is sent back to the user upon a new view of the website when update fails.
WebConFig	CurrentConFig requestCurrent() UpdatedWebConfig WebConfigUpdate() Response RevertBack()	Obtain the current web config for backup before updating. Website is benign updated with the result being passed or previous config. If failed, the config is safely reverted.

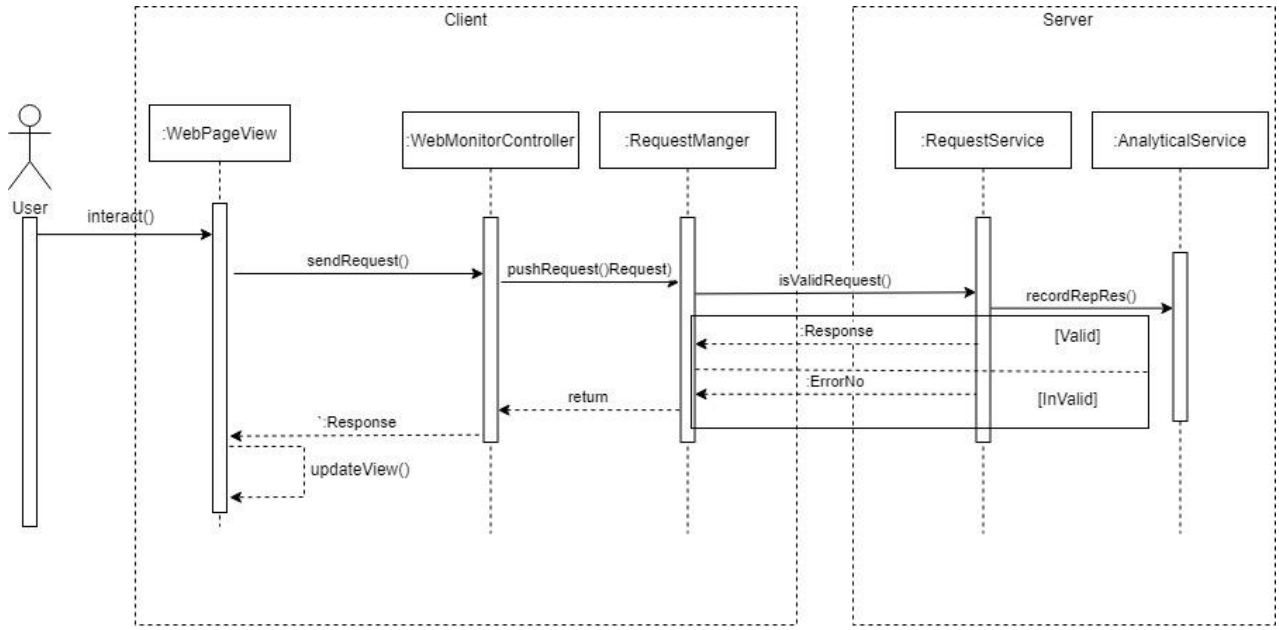

Figure 10.: Sequence diagram for UC-7

The following table defines the interacting elements of the above Sequence Diagram for UC-7.

Element	**Method**	**Description**
WebPageView	interact() updateView()	Provides user interface to interact with various website elements.
WebMonitorController	Response sendRequest()	Process uses input and users request from the server.
Request Manager	return pushRequest(request)	Sends requests that the user makes by interacting with the WebPageView.
RequestService	Response Error No isValidRequest()	Validates the request sent by the user, if valid will send back the needed response, otherwise will return an error code.
AnalyticsService	recordReqRes()	Records all request information, including if it was a valid request. Used for tracking performance and various other metrics.

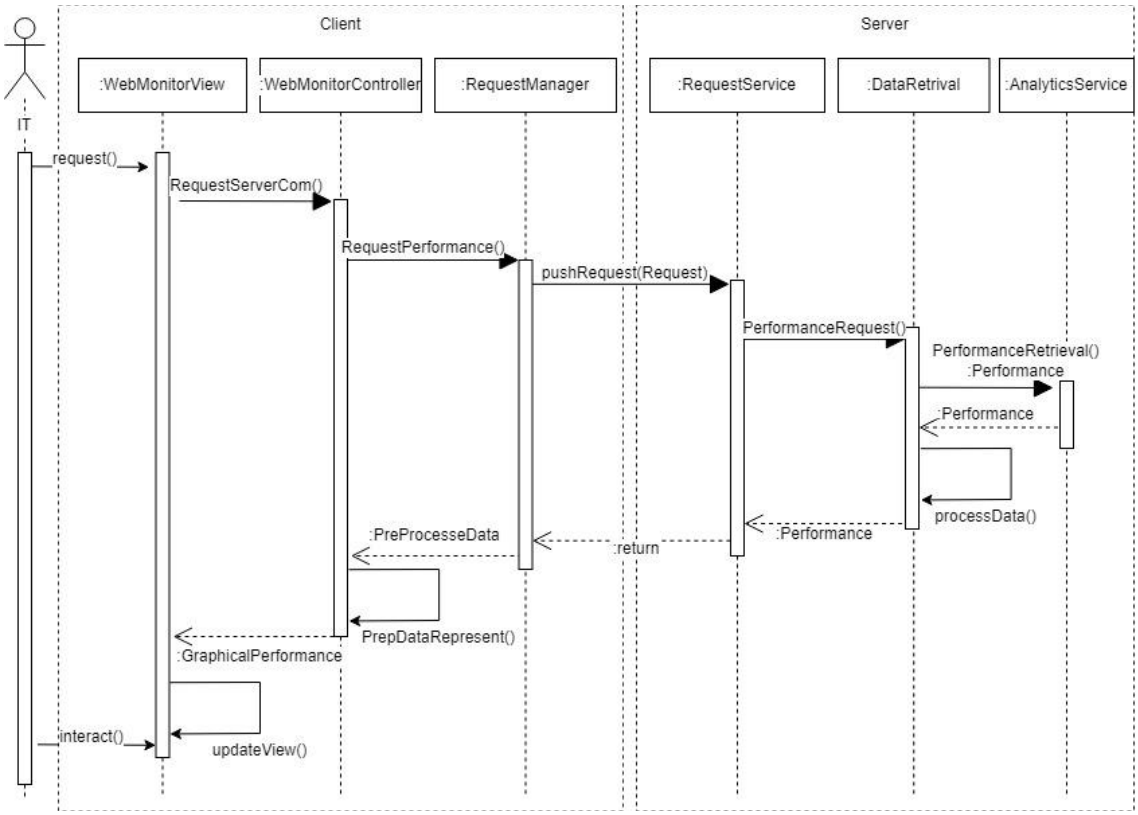

Figure 11.: Sequence diagram for UC-8

The following table defines the interacting elements of the above Sequence Diagram for UC-8.

Element	**Method**	**Description**
WebMonitorView	interact() request() updateView()	Allows IT to interact with and request the elements of the system.
WebMonitorController	GraphicalPerformance RequestServerCom() PrepDataRepresent()	Upon IT interaction a request is sent for a graphical view of the website performance.
Request Manager	PreProcessData RequestPerformance()	Allows a client to talk to the server for pre processed data of performance tobe sent back.
RequestService	Return pushRequest(Request)	The request is pushed to the server for the collected performance data to be returned back.
DataRetrival	Performance PerformanceRequest() processData()	Provide information on what information needs to be sent back as performance data while also processing raw data before sending.
AnalyticsService	Performance PerformanceRetrieval()	Responds with the correct performance data tobe sent back to the client upon retrieval request.

Step 7: Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

The following table summarizes the status of the different drivers and the decisions that were made during the iteration:

Not Addressed	Partially Addressed	Completely Addressed	Design Decisions Made during Iteration
		UC-1	The use case has been addressed within the primary use case modules
		UC-2	The use case has been addressed within the primary use case modules
		UC-5	The use case has been addressed within the primary use case modules
		UC-7	The use case has been addressed within the primary use case modules
		UC-8	The use case has been addressed within the primary use case modules
	UC-10		Modules across the layers and preliminary interfaces to support this use case have been identified
		QA-1	UC-2 has been given the proper elements that support its case.
	QA-2		All use cases have been given the proper elements that support their case.
	QA-3		Preliminary support for servers and performance put into place. UC-7 has been given the proper elements that support its case.
		QA-5	Data collection is now handled and being monitored. UC-8 has been given the proper elements that support its case.
	QA-6		UC-5 has been given the proper elements that support its case.
	CON-5		User interface modules have been identified
		CON-7	User interface modules have been identified

	CON-11		As servers have been put into place, user data can now be stored and save states can exist
CRN-3			The need for section testing of the web application was a major concern and this means block testing of different website parts is needed for web functionality testing. No more decisions to this have been made.

Iteration 3: Addressing Quality Attribute Scenario Driver

This section presents the results of the activities that are performed in the third iteration of the ADD process. For this iteration, the selected quality attributes (QA-1, QA-2, QA-3, and QA-5) have been deemed the most important, and are the drivers that will be addressed.

Step 2: Establish Iteration Goal by Selecting Drivers

For this iteration, the focus is on the QA-1, QA-2, QA-3, and QA-5 quality attributes, which are listed below in the scenario table:

QA-1 Security	The user sends the wrong info to the system during normal operation. The system must detect which connection made the incorrect attempt.
QA-2 Availability	The system will always maintain availability. After maintenance or downtime, the system must come back online within 5 to 10 seconds. Users may also create faults during login and/or website usage, an appropriate response measure must occur.
QA-3 Performance	IT and the server collect information of performance data within 2 mins for ensuring that data loss, jitter, latency, the throughput does not exist when processing out user requests. IT views performance data that only lasts for a couple of hours within 1 sec of viewing.
QA-5 Modifiability	New updates and bug fixes are being deployed to the website. The website applies said changes that are cost-effective, which should only affect what is necessary to affect.

We will be selecting all QA listed above as they are the most critical quality attribute scenarios because without availability and security, the website will not be able to function properly and provide confidence in user protection.

Step 3: Choose One or More Elements of the System to Refine

For this iteration we will want to refine the **Application Server and Database Server** as these are the two elements within the deployment plan that availability depends on and that we as architects can control.

Step 4: Choose One or More Design Concepts That Satisfy the Selected Drivers

The design concepts used in this iteration are the following:

	Design Decisions and Location	**Rationale and Assumptions**
QA-1	**Inform actors** can be introduced for informing users of unwanted account access through proper validation and tracking of failed attempts.	Since user login is required for entering the website, a way of informing users of unwanted access to their account should also be done. This provides the users with confidence of website counter measures from unwanted attacters. Further improvements can also be done through high level encryption.
	Introduce **limited access**. Done by checking user permissions within the database.	It is essential that users have pre applied permissions to their account depending on what role they are seen as. Though limiting access we can make sure that users can only see what they are allowed to see. Not limiting access can cause unwanted website changes to occur and can result with more down time to fix problems.
QA-2	Introduce the **sanity checking** tactic. We can do so by validating the outputs of our seemingly functional systems. This can also be automated for further efficiency and monitoring gains.	By checking systems we believe to already be functioning, we can ensure that our system will always remain available. This is also important as from a diagnostics perspective, the data from the system's health can come back fine, however there could be some issue where errors are not firing. Through sanity checks we can ensure availability.
	We can introduce **active redundancies** by copying critical parts of our system	By replicating the most critical parts of our system, we are less prone to complete failure as the replicated versions can always handle the extra load should some part of the system fail.
	We can introduce the **rollback** tactic through good versioning control and documentation of functioning versions of the website	Since our website will constantly be updated with new games or functionalities, having a rollback is a good fall back in the case a major bug makes it through to production. In that case, we can simply rollback to a previous version of the website and maintain availability.
QA-3	Use **active redundancy** and load balancing in the server	Since we are using active redundancy, we can distribute the load between replicated servers and balance the load in order to increase performance across all servers, instead of overloading a single server
	Introduce **Monitors** to check for data loss, jitter, and latency issues	Monitors can help maintain other parts of the system, and check in on the many different servers operating. Monitors will be able to check for any issues involving data loss, jitter, or latency, and therefore ensure that server operations run smoothly
QA-5	Introducing the **splitting module tactic**, we can do this by reducing the size of the module, which can be achieved	Since the admin constantly manges users and applies changes or modifications when necessary to fix bugs and manage updates, splitting or reducing the capacity and size of the games would help to lead a fast and successful video game website that should have much fewer issues and

	by reducing the size and capacity of more complex games that would make it much easier to handle.	some issues that occur should be quick fixes. Moreover, by using the splitting module tactic we would have a much more simple website to understand and control.
	Introducing the **Abstract common services tatic**, when two similar systems implement alike services changing the functions to a general form can prove to be more cost effective.	Since our website will constantly make updates and provide many services which may be similar such as logging in and registering. Implementing the website in a general form will not only make handling the website easier but it also proves to be more cost effective.

Step 5: Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

The instantiation design decisions are summarized in the following table:

Design Decisions and Locations	**Rational**
Inform actors with email notifications with a detector after 3 attempts in the application server.	Because the detecting system will be up at all times, it makes sense to use a detector for detecting unsuccessful login attempts.
Establish limited access for users with user role detection from database server upon login and onward.	Establishing different permissions restricts normal users from using technically under surface functionalities of the website.
Establishing active redundancy in the application server by replicating the most critical parts in the system.	Since in the server there are similar functions activated during the same time, copying the crucial parts in the system would be needed to balance the system, which should help to run a successful system. A Load-Balanced Cluster pattern is used for the redundancy of applications. This will also allow for the management of the replicates(LoadBalancer & Database). A CRN-4, for Replicate Managing is also created because of this.
Establishing a reduced size of a module, by reducing the file size of games.	Establishing a split module would help significantly by decreasing the capacity and size of games being optimized on the website. This could be done by optimizing frontend/backend, when coding. So instead of writing 500 lines of code, now only 100 lines of code would be needed to apply the same function.
Establishing abstract common services, by implementing similar updates and fixes	Since the website would require updates and fixes from time to time, it would make sense to have implement the updates and fixes using a similar method which could prove to be cost effective as we are not using an entirely different method.

Step 6: Sketch Views and Record Design Decisions

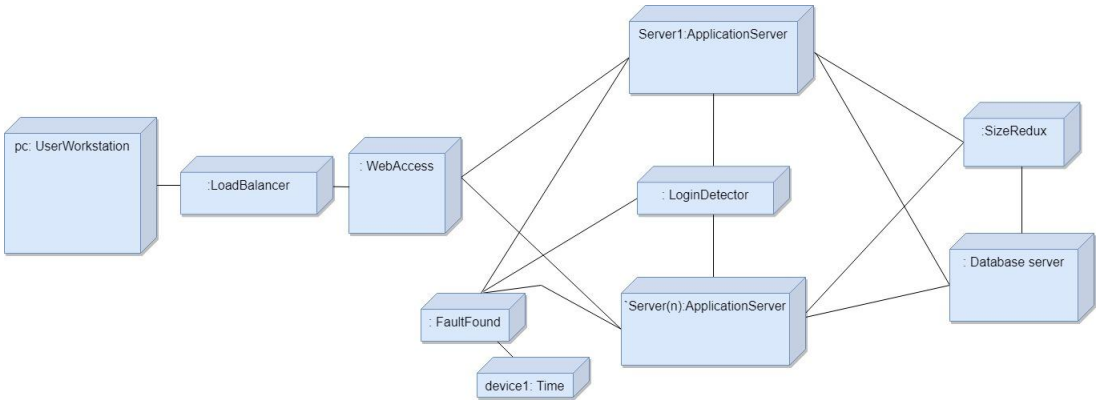

Figure 12 : Refined Deployment Diagram

The table below describes the new elements that were not derived in iteration 1.

Element	**Responsibility**
LoadBalancer	Provides load mitigation for the website when high levels of traffic occur.
WebAccess	Provides users with the correct web page when logging in and retrieves, based on user role, correct web layout.
LoginDetector	After 3 attempts of failed login attempts a notification is pushed to FaultFound with email.
FaultFound	Push identity confirmation to the email of the actual user or not for unwanted access detection and notify the server of unwanted access to an account.
SizeRedux	Provides better optimization of the website on the backend when reducing the size of games being hosted and/or imported to the website.

Step 7: Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

In this iteration, important design decisions have been made to address QA-1, QA-2, QA-3, and QA-5. The following table summarizes the status of the different drivers and the decisions that were made during the iteration:

Not Addressed	**Partially Addressed**	**Completely Addressed**	**Design Decisions Made during Iteration**
		QA-1	Introduced a notification system for unwanted access to a user's account through a FaultFound. And also provided a determination of what page shall be present to the user.
		QA-2	Introduced an availability checking system for making sure the website is up at all times. The use of load balancing with active redundancies provides less chances of failure with extra load. And a previous web state implementation provides the website with a recovery option when updates break the website.
	QA-3		Introduced active redundancy for server performance when high load is being sent. This will alleviate stress on servers. And the adding of a Monitor check for performance of the website was also added for checking the website health.
	QA-5		Introduced a splitting module tactics for added control and understanding of the website while also providing a smooth web experience. And decided on providing better general forms for better handling of the website while also lowering cost down.
	CON-3		No relevant decisions made
	CON-5		No relevant decisions made
	CON-8		No relevant decisions made
	CON-11		No relevant decisions made
	CON-13		User interface modules have been identified
	CRN-3		Identifying Block testing has been accomplished with the help of modules that are associated with it.
CRN-4			Introduced a new concern called Replicate Managing. No new decisions have been created during this point of the architecture design.