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 the modules located in the different layers defined 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	Before starting to decompose the application into its functional components it is necessary to create an initial Domain Model beforehand to identify the major entities and relationships that exist within our model. If we do not create one, a suboptimal one will emerge.
Identity Domain Objects that need that map to our primary functionalities	Each functional element of the system that is distinct must be modularized into a specific piece of the system. By doing this early in the process we can mitigate the risk that a requirement will be missing further in the design process.
Decompose the Domain Objects into specialized components	We will be turning these Domain Objects into modules that can be associated with their related layer.
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 initial domain model	Primary use case entities need to be identified and modeled, but only an initial domain model is created to accelerate this phase of the design.
Map the system use cases to domain objects	An initial identification of domain objects can be made by analyzing the system's use cases.
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

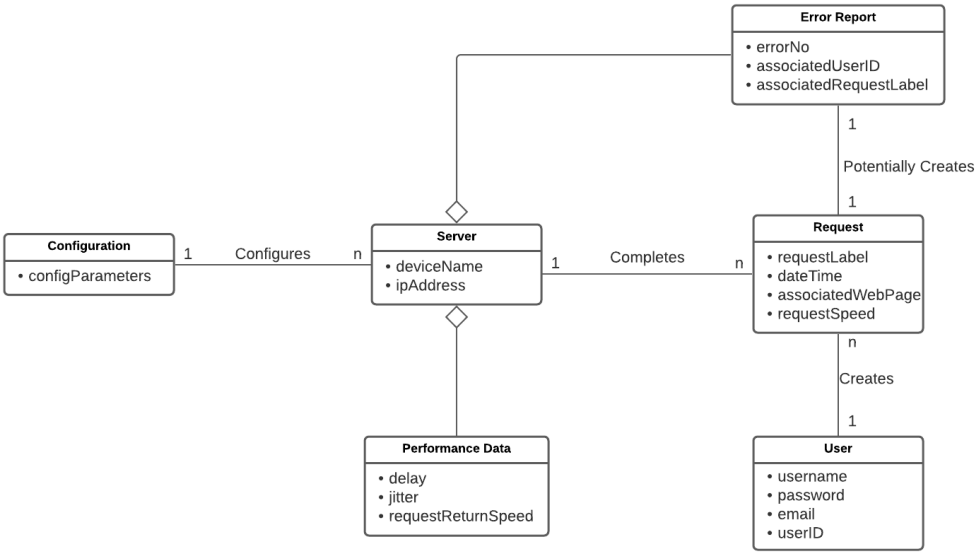

Figure 4: shows an initial domain model for the system

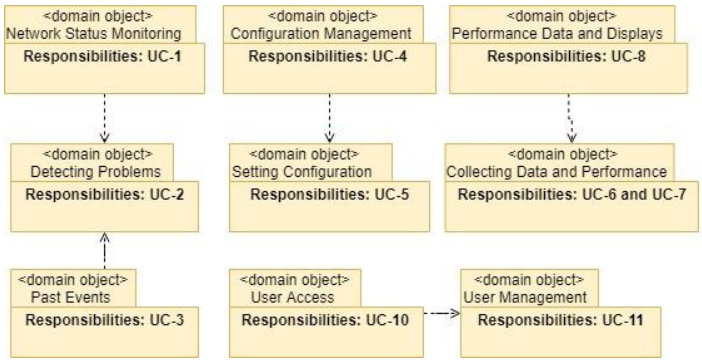

Figure 5: Domain objects associated with the use case model

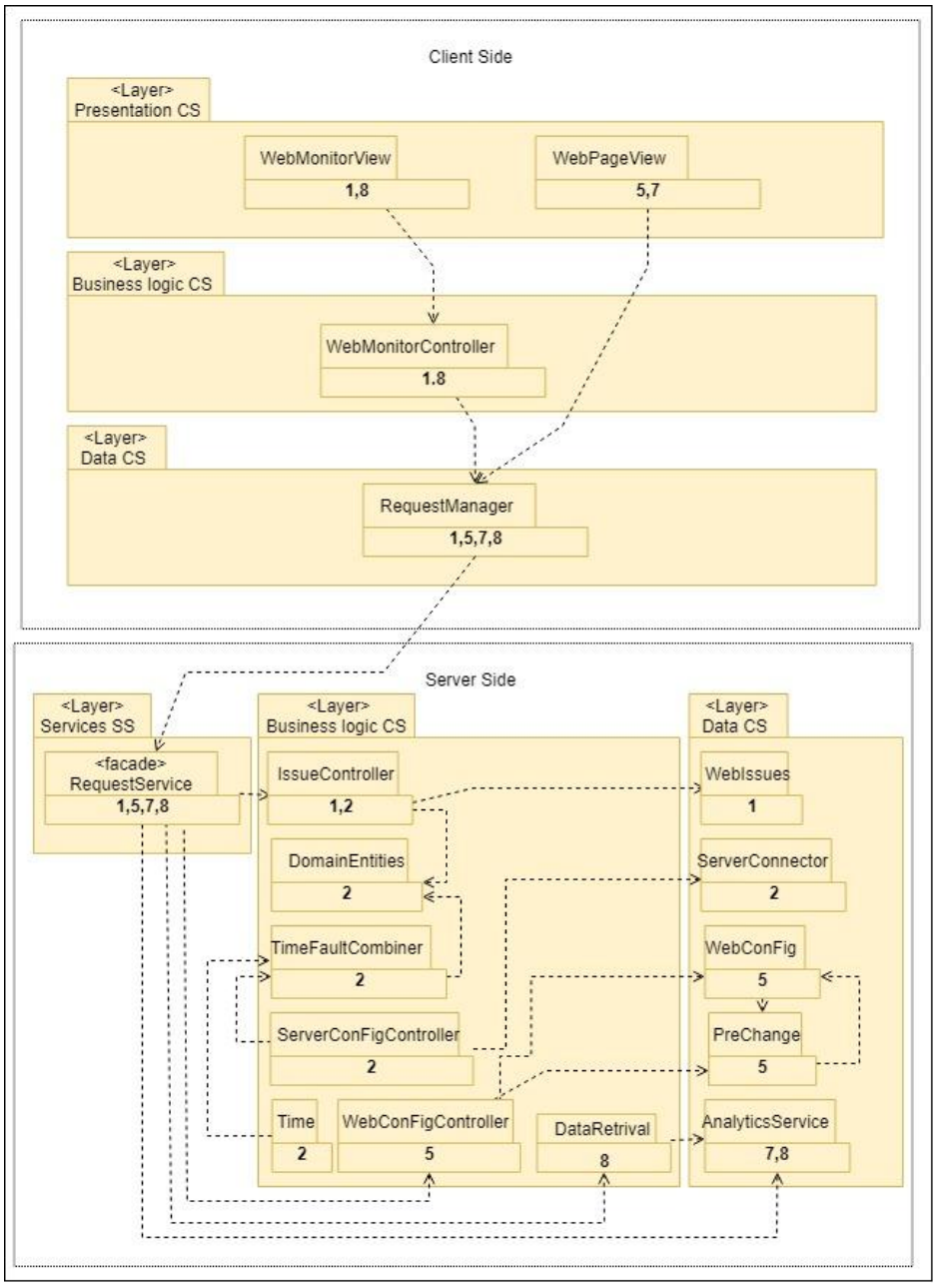

Figure 6: Primary Uses Case supported Modules

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.

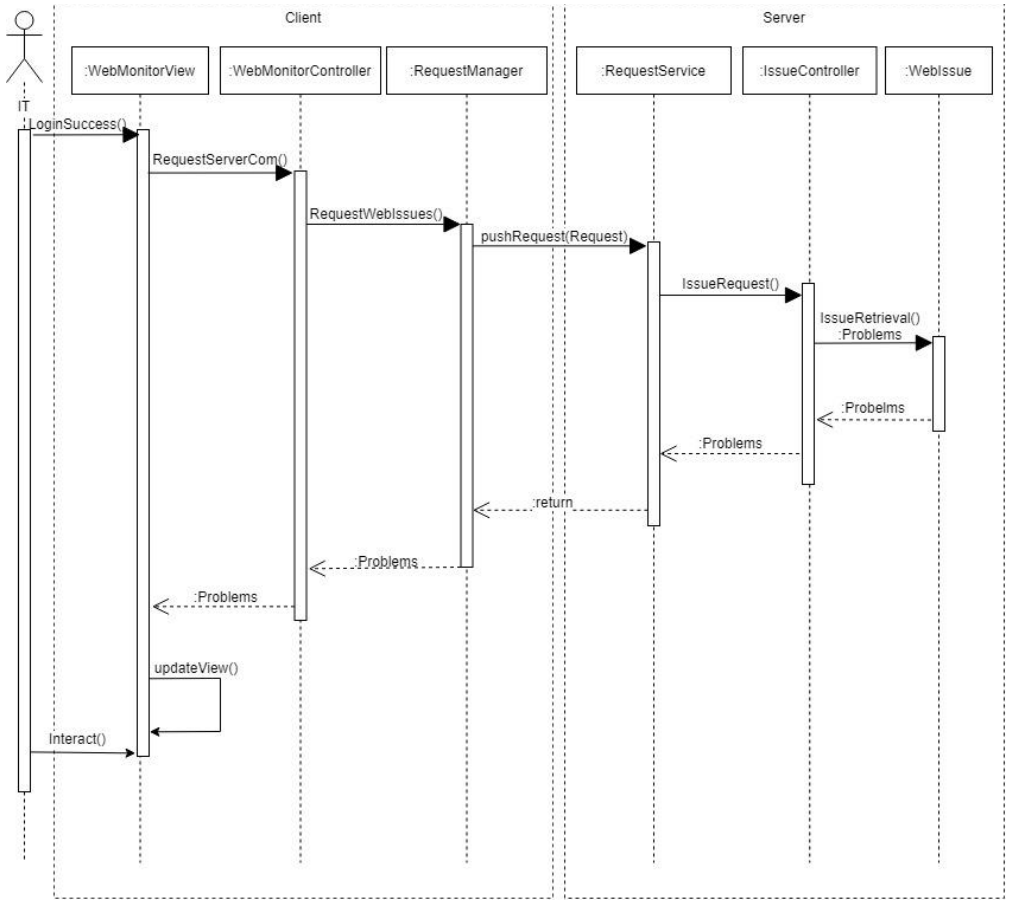

Figure 7: Sequence diagram for UC-1

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

Element	**Method**	**Description**
WebMonitorView	LoginSuccess() interact()	Upon Login success IT is provided with interactive lists of issues.
WebMonitor--Controller	Problems RequestServerCom()	Provide the user with website issues for the user to interact with.
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.

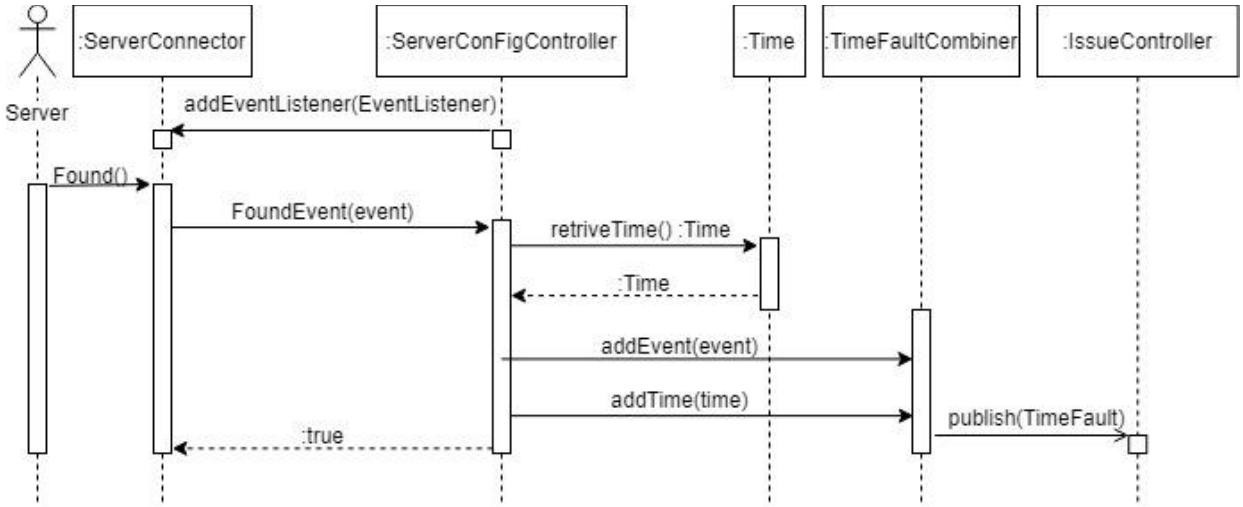

Figure 8: Sequence diagram for UC-2

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.

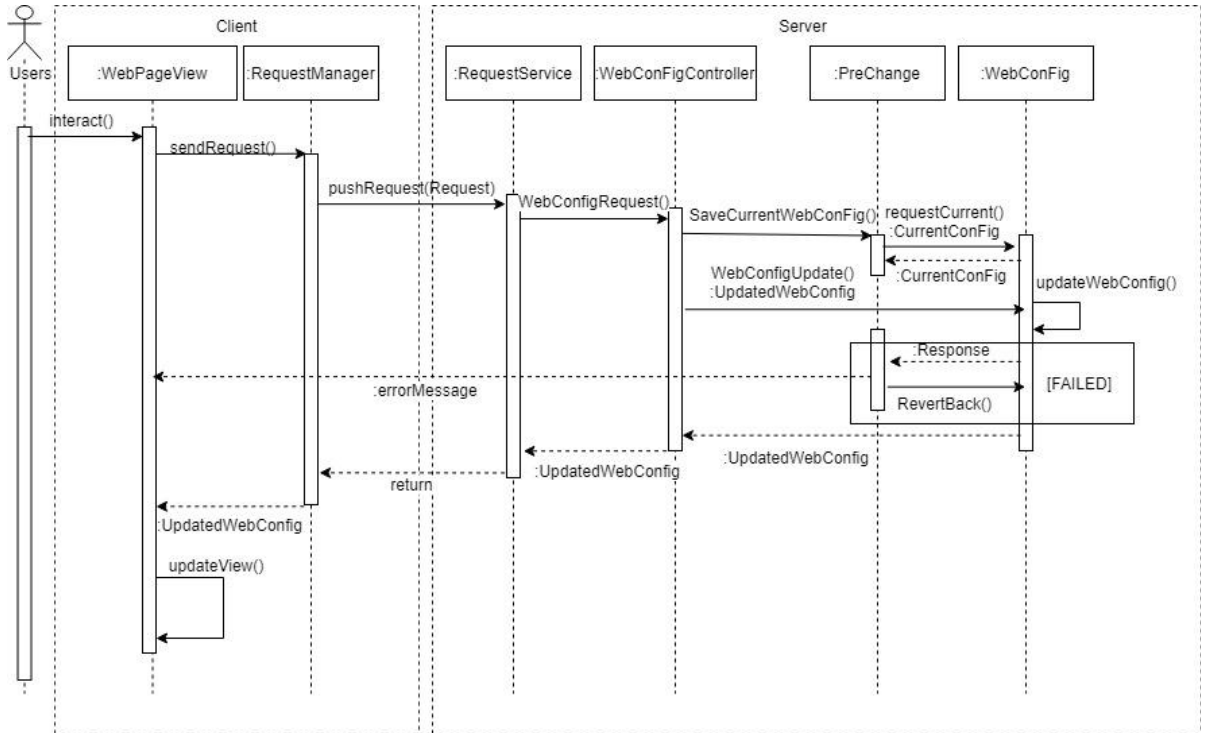

Figure 9: Sequence diagram for UC-5

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

Element	**Method**	**Description**
WebPageView	interact()	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.

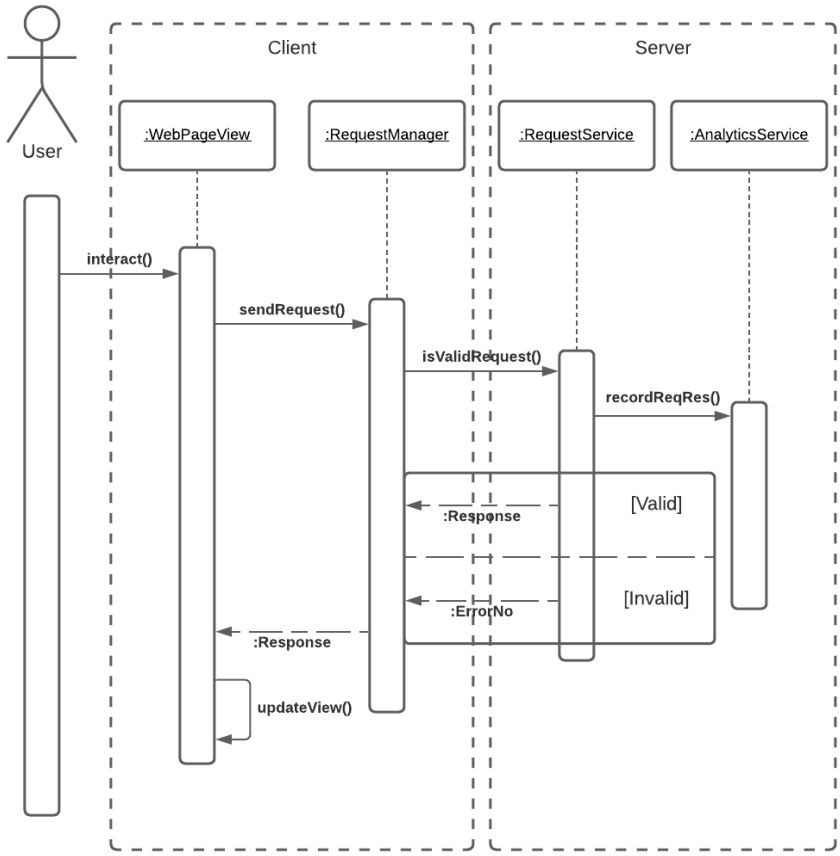

Figure 10.: Sequence diagram for UC-7

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

Element	**Method**	**Description**
WebPageView	interact()	Provides user interface to interact with various website elements.
Request Manager	Response sendRequest()	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.

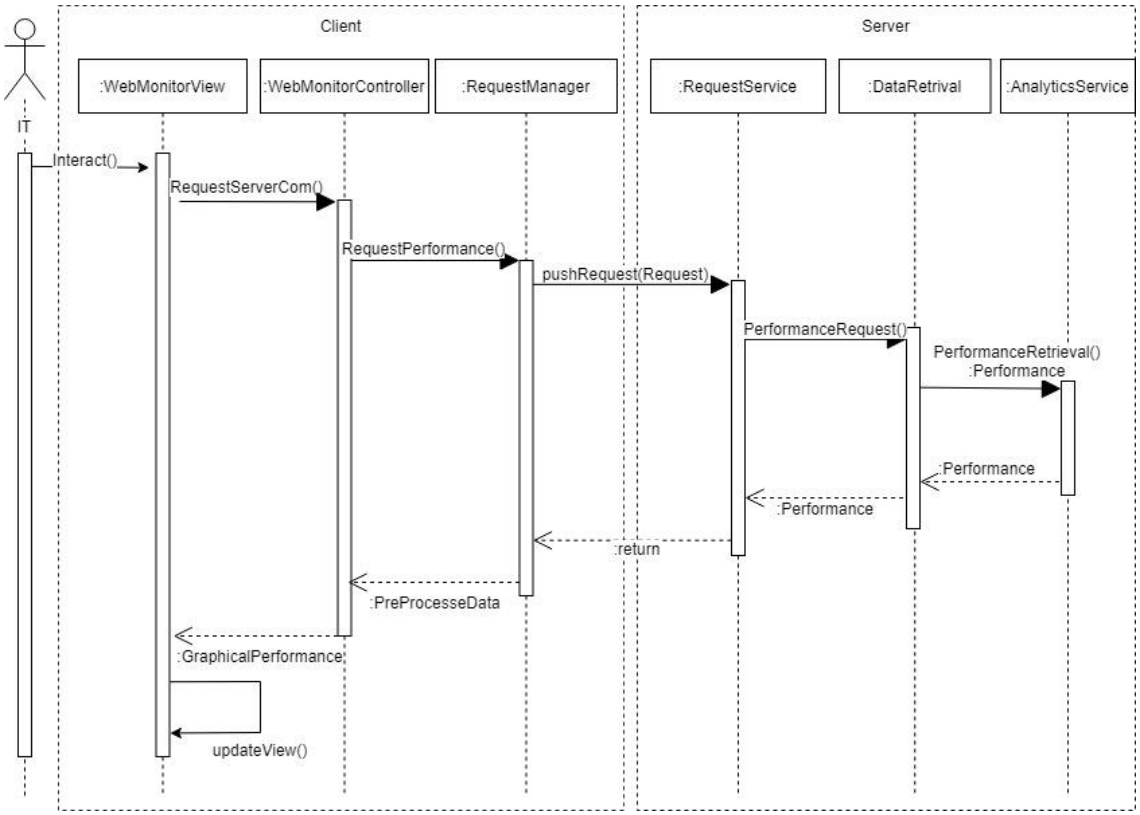

Figure 11.: Sequence diagram for UC-8

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

Element	**Method**	**Description**
WebMonitorView	interact()	Allows IT to interact with the elements of the system.
WebMonitorController	GraphicalPerformance RequestServerCom()	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()	Provide information on what information needs to be sent back as performance data.
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	Modules across the layers and preliminary frameworks to support this use case have been identified
		UC-2	Modules across the layers and preliminary frameworks to support this use case have been identified
		UC-5	Modules across the layers and preliminary frameworks to support this use case have been identified
		UC-7	Modules across the layers with preliminary data collection, creation, and visualization to support this use case have been identified
		UC-8	Modules across the layers with preliminary data collection, creation, and visualization to support this use case have been identified
	UC-10		Modules across the layers and preliminary interfaces to support this use case have been identified
		QA-1	The elements that support the associated use case (UC-2) have been identified.
	QA-2		The elements that support the associated use case (UC-1) have been identified.
	QA-3		Preliminary support for servers and performance put into place.The elements that support the associated use case (UC-7) have been identified.
		QA-5	Data collection is now handled and being monitored. The elements that support the associated use case (UC-8) have been

			identified
	QA-6		The elements that support the associated use case (UC-5) have been identified.
	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
CON-13			No relevant decisions made
CRN-3			This new architectural concern is introduced in this iteration: A majority of modules shall be unit tested. At this point, no relevant decisions have been made.