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 by reducing the size and capacity of more complex games that would make it much easier to handle.	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 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

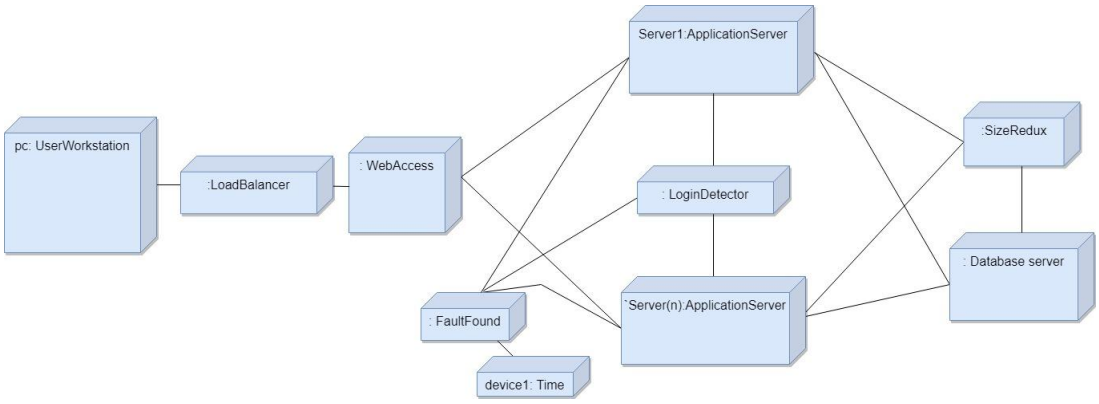

Figure 12 : Refined Deployment Diagram

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.