Task 3: Scenario analysis

* Attack vector

Please analyze the potential security concern of the above script.

var Shipcity;ShipCity = Request.form ("ShipCity");

var **SQL** = "select \* from OrdersTable where ShipCity = '" + ShipCity + "'";

(**hint: The script builds an SQL query by concatenating hard-coded strings together with a string entered by the user.**)

The above script is example of SQL injection attack. SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server.

I would correct the code like this :

SELECT ShipCity, City FROM OrdersTables;

* Cloud design pattern

You and your team are designing a web application, where web pages are populated with data retrieved from external services. Here are a few performance considerations:

* If the system implements minimal caching, most hits to each of these pages will cause a round trip to the service. If the service does not respond in this time the logic in each web page will assume that the service is unavailable and throw an exception;
* However, if the service fails and the system is very busy, users could be forced to wait for up to 60 seconds before an exception occurs. Eventually resources such as memory, connections, and threads could be exhausted;
* scaling the system by adding further web servers and implementing load balancing may delay the point at which resources become exhausted, but it will not resolve the issue because user requests will still be unresponsive and all web servers could still eventually run out of resources.

Please provide architecture design for this web application to alleviate the effects of the above problems?

Two techniques we can use are:

Cache Invalidation, particularly Cache Aside:

In this strategy, the cache works along with the database trying to reduce the hits on DB as much as possible. When the user sends a request, the system first looks for data in the cache. If data is found, then just return it to the user. The database does not need to be involved. If the data is not found in the cache, then the data is retrieved from the database, the cache is updated with this data, and then is returned to the user.

The 2nd technique is to impelment Cache eviction policies:

Least Frequently Used (LFU). Here, we need to count how often a cache item is accessed. Those that have the least use frequency are discarded first. It is a notion that the least used data are wasting space of cache. So, remove them and update the cache with fresh data.