10 Best Practices to Improve ASP. Net Core Performance

Here is a major reason why ASP.NET Core is the favorite framework. For all users, it's performance.
It offers developers a stable foundation to build web-based applications. It is crucial to check your application regularly to meet expectations. Hireasp.net is here to hire asp.net developer for your business application.
This blog is about some essential tips for improving speed and performance. It's for your ASP.NET Applications that are Core.

Use the Latest Version
Every version comes with the latest, more advanced capabilities and faster performance. When developing applications with ASP.NET Core. Make sure you use the most current version of ASP.NET Core. Since Microsoft constantly improves its performance in the most recent version over the earlier version.

1. Avoid Blocking Calls
It is important to design your ASP.NET Core application to execute several processes at once. You can accomplish this by using an Asynchronous API. That can handle thousands of requests and does not rely on calls to be blocked. Instead of waiting for an asynchronous job to finish, it will begin working in a different thread.
The issue in ASP.NET Core applications is blocking the asynchronous execution through calls to Task. The thread is blocked until the task is complete and waits for completion. Run since ASP.NET Core is already running threads within the standard thread pool. Run will result in unnecessary thread Pool scheduling.
Instead, create hot code paths that are synchronous and call I/O, data access. And patterns APIs that run for long periods to run operations asynchronously. So that they can be carried out without affecting other processes.

Ensuring that the technical aspects are in place for a running application can be difficult. We have experts in the field who could become your extended team, able to handle your projects. If you're looking to create the app from scratch or modify your existing application.

2. Use Cache

If you are looking to improve the performance of your application. The well-known method is to cut down on the number of requests made to the server each time. It works: A call is made to the server, and the response is saved. The next time a call is placed for the same response, instead of calling for the server's help. This information is compared with the stored data. And when it's found to be compatible, it is read from the database instead of calling the server. This is how caching can save time and enhance overall performance. Reducing server calls frequently. It is possible to do either server-side or server-side client-side caching. ASP.NET Core allows different types of caching, including caching in memory Response caching. And also Response caching, Distributed caching, and more.
Always make sure to use it to improve performance!

3. Optimize Data Access
To enhance the performance of the software, we need to improve the efficiency of the data access logic. Since the data is removed directly from the databases, retrieving each time takes time. Here are some methods that can increase your app's performance.

- Reduce the number of HTTP calls
- Instead of making several connections to your server obtaining the data in just one or more calls.
- Make a cache that will store unchanged data.
- Do not access the data ahead of time if it isn't needed. This improves the load on the application and can be slow.

4. Optimize Custom Code
Optimizing the code and business logic aids in enhancing the performance of your applications. Here's what you can do.

- Create a custom log and authentication or some custom handler that runs each time you request.
- Do not run custom executions long-running in the logic layer or middleware. Since it will block the server's request and increase the time to fetch the app's data. Instead, you can optimize your software either on the client or server side.
- Asynchronously complete long-running tasks to ensure that other tasks do not get affected.
- Real-time client-server communication is SignalR that works Asynchronously.

5. Use Response Caching Middleware

Create fast code by using Middleware elements that can optimize frequently-used code paths. Store responses and serve them from the cache. The component is available in the
It is recommended to employ tools to profile performance like PerfView to find hot code pathways.

6. Minimize Large Object Allocations
Although the. Net Core garbage collector is in charge of all allocations. And releases of memory on its own within the ASP. NET Core application. Cleaning of objects that have not been referenced requires CPU time. Developers should therefore avoid placing unwanted objects into hot code routes. Garbage collection can be expensive and huge heap generations like generation. That requires an indefinite suspension of app performance to clear it. A frequent allocation and cleaning could slow down the app's performance.
Tips:

- Caching larger objects because it helps avoid expensive allocations.
- Make use of an array pool to keep massive arrays.
- Do not allocate large objects to hot code routes.

Our team has a wealth of knowledge of working with Asp.Net applications. We have had a large client base with us on their side for a long time.

7. Use JSON Serialization
.NET Core 3.0 makes use of System.Text.Json to perform JSON serialization. This means that you can write and read JSON Asynchronously. with no waiting around for the other tasks to complete. Utilize JSON since it can improve the application's performance more than Newtonsoft.Json. Json namespace offers high performance and low allocation, and compatible capabilities. That Object in JSON text serialization and the deserialization process of JSON text into objects.

8. Use Response Compression
The compression of file size is a different factor that can improve performance. Response compression reduces the size of the file while in ASP.NET Core. It's available as a middleware component. The majority of responses aren't natively compressed. This is typically the case with CSS, JavaScript, HTML, XML, and JSON.

- Do not compress natively compressed files like PNG images.
- Do not compress files less than 150-1000 bytes.

9. Minimize Exceptions

The process of throwing and catching exceptions can be slower than other code patterns. Thus they shouldn't be used infrequently and should not be used to regulate the normal flow of programs.
Tips:

- Create code logic to identify and deal with situations that can cause exceptions.
- Throw or catch exceptions for unpredictability or unusual circumstances.

You can utilize App diagnostic tools like Application Insights to identify common errors in apps and how they work.

Excellent Tips to Improve Performance

- Prefer ReadFormAsync over Request Form
- Do not store IHttpContextAccessor.HttpContext in a field
- Do not open HttpContext through multiple threads. Since it's not secure and may cause undefined behavior. Like crashes, hangs, or data corruption.
- Do not record services that are injected into controllers via background threads.
- Do not alter the headers or status code after the HTTP response body is started. Because ASP.NET Core doesn't buffer the HTTP response body.

Client-Side Improvements

- **Bundling and Minification**

Optimize your content before loading it using minification. Make sure to reduce the code and load all the client-side assets. After you have reduced the files, put them together to load faster. Like it is with the zip file. Then compress it, and zip it.

- **Load JavaScript at Last**

Be sure that you load JavaScript files last. With this method, your website will be completed in a short amount of time. Also, when JavaScript is running, DOM elements are already there. Therefore, your application can be enhanced speedier.

- **Shrink Images**

Large images require a lot of effort to load. Instead, you can reduce the images with compression tools to reduce the loading speed.

- **Use CDN**

If you're using third-party libraries to CSS and JavaScript that comes with a CDN version. You should try to use your CDN file path instead of downloading the library files. CDN has multiple servers in different zones. If you're using a site within a single region, it will download the CDN file through the server. When accessing the site, which is quicker than self-loading the large library file.