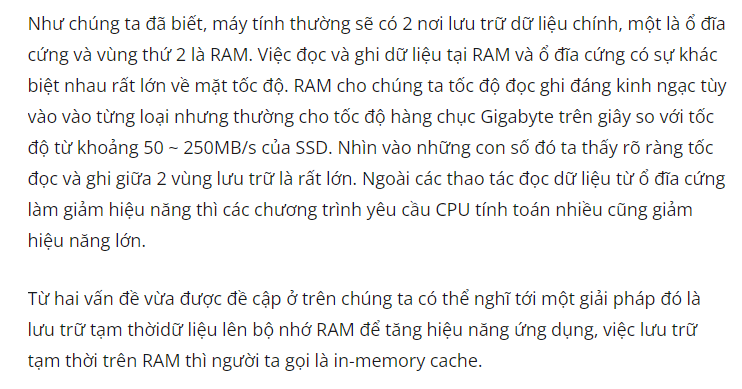
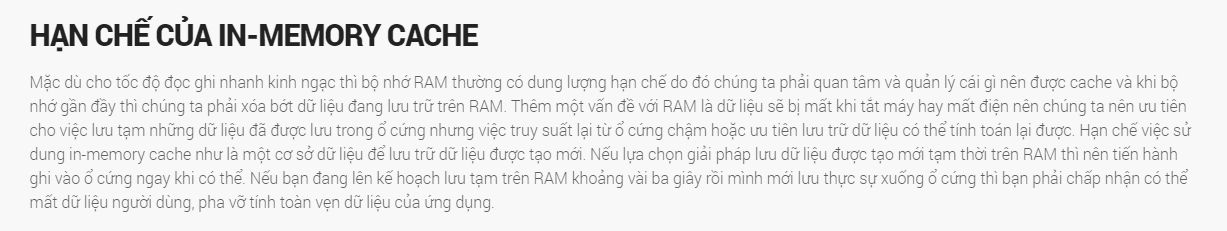
ASP.NET CORE PERFORMANCE BEST PRACTICES

(Các phương pháp hay nhất về hiệu suất của ASP.NET)

1. **Sử dụng các kỹ thuật caching**

* In-memory caching (cache dữ liệu trong bộ nhớ máy chủ web)





* Distributed cache (cache bằng việc lưu dữ liệu cache bên ngoài hệ thống ví dụ Redis là 1 trong số loại này)

Redis là một mã nguồn mở được dùng để lưu trữ dữ liệu có cấu trúc, có thể sử dụng như một database, bộ nhớ cache hay một message broker.

1. **Sử dụng DbContextPooling để tăng hiệu năng cho ứng dụng .NET Core**

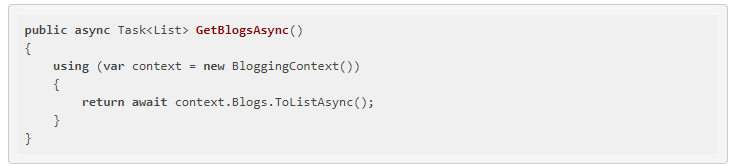
* Trường hợp của **AddDbContext,** một thể hiện của nó được tạo cho mỗi request và sẽ tự hủy khi công việc xong**.** Bạn có thể thấy là cứ mỗi một request đều có một đối tượng DbContext được tạo.
* Nó vẫn chạy tốt nếu chỉ có khoảng 1000 request 1 lúc, vậy sẽ có 1000 đối tượng được tạo và hủy, trừ khi bạn dùng Singleton. Tạo và hủy quá nhiều đối tượng sẽ ảnh hưởng đến performance.
* Nói một cách đơn giản thì **DbContextPooling** – **là một bể chứa các đối tượng có thể được sử dụng lại** được tạo ra. Thay vì mỗi lần chúng ta phải tạo mới đối tượng cho mỗi request, đầu tiên hệ thống sẽ kiểm tra xem có đối tượng nào có sẵn trong pool không?
* Bạn có thể thấy ở hình trên, nếu có bất kỳ context nào có sẵn thì các context đó sẽ được dùng thay vì tạo mới các thể hiện mới. Điều này sẽ giảm số lượng context được tạo mới trong mỗi request.
* Cũng thay vì hủy tất cả các thể hiện của context, nó được trả về cho pool và reset trạng thái của thể hiện về mặc định. Vì thế các thể hiện của context có thể được dùng trong tương lai.

Link tham khảo: <https://tedu.com.vn/kien-thuc/su-dung-dbcontextpooling-de-tang-hieu-nang-cho-ung-dung-net-core-273.html>

* Cảnh báo: Tránh sử dụng DbContext Pooling nếu bạn duy trì trạng thái của riêng mình (ví dụ: các trường riêng tư) trong lớp DbContext dẫn xuất của bạn mà không nên được chia sẻ qua các yêu cầu. EF Core sẽ chỉ đặt lại trạng thái mà nó biết trước khi thêm một cá thể DbContext vào nhóm.

1. **Tránh gọi đồng bộ mà nên sử dụng bất đồng bộ (lưu dữ liệu bất đồng bộ và truy vấn bất đồng bộ)**

* Cố gắng tránh các lệnh gói đồng bộ (synchronous) khi phát triển ứng dụng ASP.NET Core 3. Lệnh gọi đồng bộ sẽ block các lệnh thực hiện sau nó cho đến khi nó xử lý xong. Khi lấy dữ liệu từ API hoặc thực hiện một tác vụ như là I/O (input/output) tốn thời gian thì nên xử lý nó theo cách bất đồng bộ.
* Tránh dùng **Task.Wait** và **Task.Result**, và nên sử dụng từ khóa **await**. Đoạn code sau là ví dụ.
* **Truy vấn bất đồng** bộ tránh việc bị block một thread trong khi thực thi câu lệnh vào database. Truy vấn bất đồng bộ thì ưu tiên nhanh để có trải nghiệm người dùng tốt. (Ví dụ: sử dụng **ToListAsync(), ToArrrayAsync, SingleAsync()**).



* **Lưu dữ liệu bất đồng bộ** tránh việc block thread khi các thay đổi được lưu vào database. EF Core cung cấp phương thức DbContext.SaveChangesAsync() như một lệnh lưu thay vì DbContext.SaveChanges()

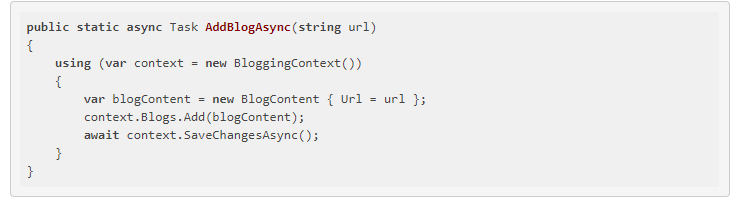
***Đề xuất:***

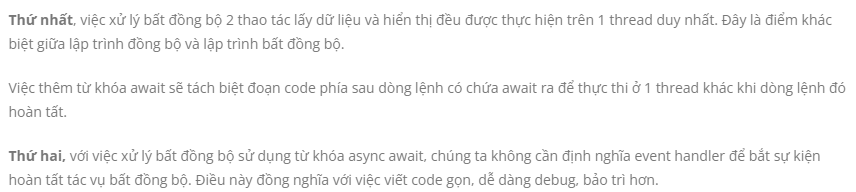
***- Truy cập toàn bộ bằng lệnh gọi bất đồng bộ***

***- Chỉ lấy ra những gì cần thiết***

***- Nên sử dụng no-tracking trong Entity Framewỏk Core khi lấy ra chỉ với mục đích read-only***

***- Sử dụng các lệnh lọc và tổ hợp trong LINQ (với .Where,.Select hay .Sum), đảm bảo rằng nó thực thi trong database.***



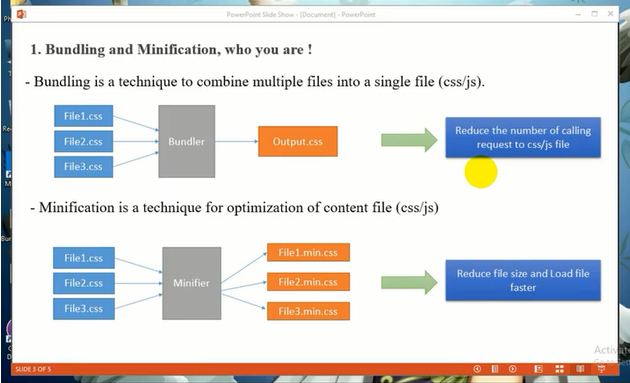


1. **Sử dụng thư viện làm việc với JSON mới**

* ASP.NET Core 3.0 sử dụng [**System.Text.Json**](https://docs.microsoft.com/dotnet/api/system.text.json) làm thư viện mặc định khi thao tác với JSON. Chúng ta có thể đọc ghi JSON một cách bất đồng bộ. Nó tối ưu tốc độ hơn **Newtonsoft.Json**. Namespace **System.Text.Json** cung cấp các tính năng làm việc với JSON như sau:
* Hiệu năng cao
* Sử dụng ít tài nguyên
* Tương thích
* Chuyển Object sang JSON và ngược lại

1. **Giảm số lượng HTTP Request**

* Giảm số lượng HTTP Request là một trong số các tối ưu chính. Cache trang web và tránh chuyển hướng ở client là cách để giảm số lượng kết nối đến web server.
* Sử dụng các kỹ thuật sau để giảm HTTP Request:
* 1. Dùng minification: Minification loại bỏ các ký tự không cần thiết trong code mà không thay đổi tính năng, nó cũng giảm kích thước file. Sau khi áp dụng minification, các tên biến sẽ ngắn hơn thành một ký tự và các comment cũng như khoảng trắng không cần thiết sẽ bị loại bỏ.
* 2. Dùng bundling: Bundling kết hợp nhiều file vào một file duy nhất, giảm số lượng request. Bạn có thể có nhiều bundle trong một trang web.



Link cài đặt và cấu hình: <http://dotnetvn.com/Lam-viec-voi-Bundling-va-Minification-trong-AspNet-Core-p115.html>

* 3. Dùng các hình ảnh gộp (sprite image)

1. **Sử dụng Exception khi thật cần thiết**

* Các exception thường ít xảy ra. Throw và catch exception sẽ tốn tương đối thời gian trong code
* Đừng throw và catch exception trong code flow
* Sử dụng exception khi thật cần thiết
* Nên dùng Global Exception handler

1. **Tải file JavaScript cuối cùng**

* Tải các file JavaScript sẽ được đặt xuống cuối. Nếu làm như vậy các nội dung tĩnh sẽ hiển thị nhanh hơn, người dùng sẽ không phải đợi

1. **Lấy dữ liệu cần thiết**

* Chỉ lấy những dữ liệu cần thiết – tránh lấy toàn bộ dữ liệu để tránh dư thừa và ảnh hưởng tới performance của ứng dụng