HABIB UL REHMAN

01

ASP.NET CORE

WEB API

BEST PRACTICES



@HABIBDEVELOPER

FINCHSHIP.COM

Use Attribute Routing for Clear Endpoints

Employ attribute routing for Web API controllers to define clear and concise endpoint routes.

DTOs for Request and Response Models

Utilize Data Transfer Objects (DTOs) to represent request and response models. This promotes clarity and separation of concerns.

```
public class CreateProductDto

public string Name { get; set; }

public decimal Price { get; set; }

fublic decimal Price { get; set; }

fublic IActionResult CreateProduct([FromBody] CreateProductDto productDto)

fublic IActionResult CreateProduct([FromBody] CreateProductDto productDto productDto)

fublic IActionResult CreateProduct([FromBody] CreateProductDto productDto productD
```

AutoMapper for mapping DTOs

A mapper automates the conversion process between DTOs and domain models, making your code cleaner.

```
public class CreateProductDto
{
   public string Name { get; set; }
   public decimal Price { get; set; }
}
public class CreateProduct
{
   public string Name { get; set; }
   public decimal Price { get; set; }
}
```

```
private readonly IMapper _mapper;
public ProductController(IMapper mapper)
{
    _mapper = mapper;
}
[HttpPost]
public IActionResult CreateProduct([FromBody]CreateProductDto productDto)
{
    CreateProduct product = _mapper.Map<CreateProduct>(productDto);
    //Create product logic
}
```

Versioning for API Longevity

Implement API versioning to ensure backward compatibility and facilitate future updates.

```
1 services.AddApiVersioning(options =>
2 {
3     options.DefaultApiVersion = new ApiVersion(1, 0);
4     options.AssumeDefaultVersionWhenUnspecified = true;
5     options.ReportApiVersions = true;
6 });
```

Use Dependency Injection for Services

Leverage Dependency Injection to inject services into controllers, promoting modularity and testability.

```
public class ProductsController : ControllerBase

2 {
    private readonly IProductService _productService;

4    public ProductsController(IProductService productService)
6    {
        _productService = productService;
8    }
9 }
```

Handle Errors Gracefully

Implement a consistent error handling strategy using status codes and meaningful error messages.

```
1 [HttpGet("{id}")]
 2 public IActionResult GetProduct(int id)
 3 {
       var product = _productService.GetProduct(id);
 4
 5
       if (product = null)
 6
       {
 7
           return NotFound($"Product with ID {id} not found.");
 8
 9
10
       return Ok(product);
11
12 }
```

Implement Caching for Performance

Apply caching strategies to enhance API performance, especially for data that doesn't change frequently.

```
1 [HttpGet]
2 [ResponseCache(Duration = 60)]
3 public IActionResult GetProducts()
4 {
5  // Return products with caching
6 }
```

HABIB UL REHMAN

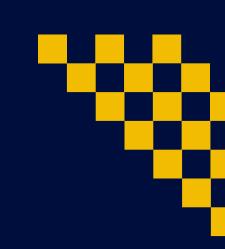


REPOST THIS POST IF YOU FIND IT USEFUL





@HABIBDEVELOPER



FINCHSHIP.COM