WEB API TIPS & TRICKS (PART1)

Welcome to Episode 1 of our journey towards writing clean code in Web API development.

In this installment, we'll lay the foundation for creating well-organized and maintainable code that will enhance the quality of your API.

Whether you're a seasoned developer or just starting out, this episode is designed to provide you with easy-to-understand tips and tricks to kickstart your clean code journey.

So, let's dive in and discover how you can craft elegant and efficient code in your Web API projects.

KHORGAMI

FOLLOW THE (SRP)

Keep Actions Focused and Concise: Follow the **Single Responsibility Principle** and ensure that each action has a single purpose.

Keep your actions concise and focused on handling specific requests. Avoid placing too much logic within a single action.

```
[HttpGet("{id}")]
public IActionResult GetProduct(int id)
{
    // Logic to retrieve a product by ID
    var product = _productService.GetProductById(id);
    return Ok(product);
}
```

MEANINGFUL AND CONSISTENT NAMING

Use Meaningful and Consistent Naming Conventions: Choose descriptive and meaningful names for variables, functions, and classes.
Follow established naming conventions like PascalCase for classes and methods, camelCase for variables and parameters, and use clear and concise names that accurately represent their purpose.

```
[HttpGet("{id}")]
public IActionResult GetProductById(int id)
{
    // Logic to retrieve a product by ID
}

[HttpPost]
public IActionResult CreateProduct([FromBody] ProductDto productDto)
{
    // Logic to create a new product
}
```

ERROR HANDLING

Implement Proper Error Handling: Handle exceptions gracefully and consistently throughout your code.

Use try-catch blocks (or middleware ...) to catch and handle exceptions at the appropriate level of abstraction.

Avoid catching general exceptions unless necessary, as it can make debugging and troubleshooting more difficult.

```
[HttpGet("{id}")]
public IActionResult GetProduct(int id)
{
    try
    {
        var product = _productService.GetProductById(id);
        if (product == null)
        {
            return NotFound();
        }
        return Ok(product);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "An error occurred while retrieving the product.");
        return StatusCode(500, "An unexpected error occurred.");
    }
}
```

PROPER INPUT VALIDATION

Implement Proper Input Validation: Validate input data to ensure that it meets the required criteria.

Use data annotations or custom validation logic to enforce validation rules on the incoming request data.

```
[HttpPost]
public IActionResult CreateProduct([FromBody] ProductDto productDto)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    // Logic to create a new product
}
```

USE DTOS FOR DATA TRANSFER

Use DTOs for Data Transfer: Use Data Transfer Objects (DTOs) to transfer data between the client and the API.

DTOs allow for better decoupling of the API from the underlying data model and provide a clear contract for data exchange.

```
public record ProductDto(string Name, decimal Price, decimal Discount)
{
    public decimal FinalPrice => Price - (Price * Discount);
}
```

CONTINUE YOUR LEARNING JOURNEY!

Thank you for joining us on this incredible journey through the Tick and Trip series on Web Api!

We hope you found these tutorials helpful and insightful.

Don't miss out on the next exciting tutorials and valuable content!

Follow us on social media to stay connected, get updates, and continue your learning journey.

We look forward to sharing more knowledge with you!

Follow us:

- LinkedIn: KHORGAMI
- medium: @Khourgami

Stay tuned for more tutorials, tips, and tricks!

Vahid Khorgami