Cheat Sheet for comprensive ASP.NET

Comprehensive ASP.NET Cheat Sheet

1. Introduction to ASP.NET

ASP.NET is a web framework developed by Microsoft for building web applications, APIs, and services. It supports multiple platforms and is highly scalable.

Key Features:

- **Cross-Platform**: Runs on Windows, Linux, and macOS.
- **High Performance**: Optimized for speed and efficiency.
- **Open Source**: Community-driven with extensive documentation.
- **Extensible**: Supports custom middleware, filters, and extensions.

---

2. ASP.NET Core vs. ASP.NET Framework

Feature	ASP.NET Core	ASP.NET Framework
Platform	Cross-Platform	Windows Only
Performance	High	Good
Modularity	Highly modular	Less modular
Dependency Injection	Built-in	Requires third-party libraries
Open Source	Yes	No
Target Framework	.NET Core, .NET 5/6/7	.NET Framework

---

3. Project Structure

Typical ASP.NET Core Project Structure:

```
/MyProject
|-- /Controllers
|-- /Models
|-- /Views
|-- /wwwroot
|-- appsettings.json
|-- Program.cs
|-- Startup.cs
|-- MyProject.csproj
```

Key Files:

- **Program.cs**: Entry point of the application.
- **Startup.cs**: Configures services and middleware.
- **appsettings.json**: Application configuration settings.
- **wwwroot**: Static files (CSS, JS, images).

---

4. Configuration

Configuration Sources:

- **appsettings.json**: Default configuration file.
- **Environment Variables**: Overrides settings based on environment.
- **Command-Line Arguments**: Can be used to set configuration values.

Example:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection":
"Server=myServer;Database=myDb;User=myUser;Password=myPassword;"
  }
}
```

Accessing Configuration:

```
var configuration = new ConfigurationBuilder()
    .AddJsonFile("appsettings.json")
    .Build();
```

```
var connectionString =
configuration.GetConnectionString("DefaultConnection");
```

---

5. Middleware

Middleware Pipeline:

- Middleware components are executed in the order they are added.
- Each middleware can modify the request or response.

Example:

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseStaticFiles();
    app.UseRouting();
    app.UseAuthentication();
    app.UseAuthorization();
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}
```

Common Middleware:

- **UseStaticFiles**: Serves static files.

- **UseRouting**: Enables routing.

- **UseAuthentication**: Handles authentication.

- **UseAuthorization**: Handles authorization.

---

6. Routing

Attribute Routing:

```
[Route("api/[controller]")]
public class ProductsController : ControllerBase
{
    [HttpGet("{id}")]
    public IActionResult GetProduct(int id)
    {
        // Implementation
    }
}
```

Conventional Routing:

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

Route Constraints:

```
[HttpGet("product/{id:int}")]
public IActionResult GetProduct(int id)
{
    // Implementation
}
```

____

7. Controllers

Creating a Controller:

```
[ApiController]
[Route("api/[controller]")]
public class ProductsController : ControllerBase
{
    [HttpGet]
    public IActionResult GetProducts()
    {
        // Implementation
    }
}
```

Action Results:

- **Ok**: Returns 200 OK.
- **NotFound**: Returns 404 Not Found.
- **BadRequest**: Returns 400 Bad Request.
- **Created**: Returns 201 Created.

Example:
```
[HttpPost]
public IActionResult CreateProduct([FromBody] Product product)
{
    if (product == null)
    {
        return BadRequest();
    }
    // Save product
    return CreatedAtAction(nameof(GetProduct), new { id = product.Id },
product);
}
```

---

8. Views

Razor Syntax:

- **@**: Used to embed C# code.
- **@model**: Specifies the model type.
- **@if, @foreach, @for, @while**: Control flow statements.

Example:
```
@model List<Product>

<ul>
    @foreach (var product in Model)
    {
        <li>@product.Name</li>
    }
</ul>
```

Partial Views:
```
@await Html.PartialAsync("_ProductPartial", Model.Products)
```

Layout Pages:
```
@{
    Layout = "_Layout";
}
```

____

9. Models

Data Annotations:

```
public class Product
{
    [Required]
    public int Id { get; set; }

    [StringLength(100)]
    public string Name { get; set; }

    [Range(0, 1000)]
    public decimal Price { get; set; }
}
```

Model Binding:

```
public IActionResult CreateProduct([FromBody] Product product)
{
    // Implementation
}
```

Validation:

```
if (!ModelState.IsValid)
{
    return BadRequest(ModelState);
}
```

____

10. Data Access

Entity Framework Core:

```
public class ApplicationDbContext : DbContext
{
    public DbSet<Product> Products { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {
        optionsBuilder.UseSqlServer("DefaultConnection");
    }
}
```

CRUD Operations:

```
using (var context = new ApplicationDbContext())
{
    var product = new Product { Name = "Laptop", Price = 999 };
    context.Products.Add(product);
    context.SaveChanges();
}
```

Migrations:

```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

____

11. Authentication and Authorization

Authentication:

```
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new
TokenValidationParameters
        {
            // Configuration
        };
    });
```

Authorization:

```
[Authorize]
public class ProductsController : ControllerBase
{
    // Implementation
}
```

Roles:

```
[Authorize(Roles = "Admin")]
public IActionResult DeleteProduct(int id)
```

```
{
    // Implementation
}
```

____

12. Dependency Injection

Registering Services:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddScoped<IProductService, ProductService>();
}
```

Injecting Services:

```
public class ProductsController : ControllerBase
{
    private readonly IProductService _productService;

    public ProductsController(IProductService productService)
    {
        _productService = productService;
    }
}
```

Service Lifetimes:

- **Transient**: New instance each time.
- **Scoped**: New instance per request.
- **Singleton**: Single instance for the application.

____

13. Logging

Configuring Logging:

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
        {
            logging.ClearProviders();
            logging.AddConsole();
        })
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.UseStartup<Startup>();
        });
```

Logging Levels:

- **Trace**: Very detailed logs.
- **Debug**: Debugging information.
- **Information**: General information.
- **Warning**: Warnings.
- **Error**: Errors.
- **Critical**: Critical errors.

Example:

```
_logger.LogInformation("Product created with ID: {ProductId}",
product.Id);
```

____

14. Testing

Unit Testing:

```
[Fact]
public void GetProduct_ReturnsProduct()
{
    var productService = new ProductService();
    var product = productService.GetProduct(1);
    Assert.NotNull(product);
}
```

Integration Testing:

```
public class ProductsControllerTests :
IClassFixture<WebApplicationFactory<Startup>>
{
    private readonly HttpClient _client;

    public ProductsControllerTests(WebApplicationFactory<Startup>
factory)
    {
        _client = factory.CreateClient();
    }

    [Fact]
    public async Task GetProduct_ReturnsProduct()
    {
        var response = await _client.GetAsync("/api/products/1");
        response.EnsureSuccessStatusCode();
        var product = await response.Content.ReadAsAsync<Product>();
        Assert.NotNull(product);
    }
}
```

---

15. Deployment

Publishing:

```
dotnet publish -c Release -o ./publish
```

Deployment Options:

- **IIS**: Windows-based web server.

- **Kestrel**: Cross-platform web server.

- **Docker**: Containerized deployment.

- **Azure**: Cloud deployment.

Example:

```
dotnet publish -c Release -o ./publish
scp -r ./publish user@server:/var/www/myapp
```

---

16. Tips and Tricks

Debugging:

- Use `Debugger.Break()` to pause execution.
- Set breakpoints in Visual Studio.

Performance:

- Use `async` and `await` for I/O operations.
- Minimize the use of blocking calls.

Security:

- Use HTTPS.
- Validate input.
- Use parameterized queries to prevent SQL injection.

Best Practices:

- Follow SOLID principles.
- Use dependency injection.
- Write unit tests.

This cheat sheet provides a comprehensive overview of ASP.NET, covering essential features, shortcuts, tips, and tricks. Use this as a quick reference guide for your ASP.NET projects.

By Ahmed Baheeg Khorshid

ver 1.0