Cheat Sheet for comprehensive ASP.NET

ASP.NET Core Overview

- **What is ASP.NET Core?**

- Cross-platform, high-performance, open-source framework for building modern, cloud-based, and internet-connected applications.
- Supports multiple platforms: Windows, macOS, and Linux.

- **Key Features**

- **Modular Design**: Built on a modular architecture, allowing developers to include only the features they need.

- **Dependency Injection**: Built-in support for dependency injection.

- **Cross-Platform**: Runs on Windows, macOS, and Linux.

- **High Performance**: Optimized for performance and scalability.

- **Open Source**: Fully open-source with a large community.

Project Structure

- **Typical Project Structure**

- `Program.cs`: Entry point of the application.
- `Startup.cs`: Configures services and the app's request pipeline.
- `appsettings.json`: Configuration settings.
- `Controllers/`: Contains MVC controllers.
- `Views/`: Contains Razor views.
- `Models/`: Contains data models.
- `wwwroot/`: Static files (CSS, JS, images).

Configuration

- **Configuration Files**

- `appsettings.json`: Main configuration file.
- `appsettings.{Environment}.json`: Environment-specific settings.
- `secrets.json`: User secrets for development.

- **Environment Variables**

- Access via `IConfiguration` interface.
- Example: `var value = Configuration["KeyName"];`

Dependency Injection

- Registering Services

- In `Startup.cs`:

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

- Injecting Services

- In a controller:

```
public class MyController : Controller
{
    private readonly IMyService _myService;

    public MyController(IMyService myService)
    {
        _myService = myService;
    }
}
```

Middleware

- Middleware Pipeline

- Configure in `Startup.cs`:

```
    public void Configure(IApplicationBuilder app, IHostingEnvironment
env)
    {
        app.UseStaticFiles();
        app.UseAuthentication();
        app.UseMvc();
    }
```

- Custom Middleware

- Create a middleware class:

```
public class MyMiddleware
{
    private readonly RequestDelegate _next;
```

```
    public MyMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task InvokeAsync(HttpContext context)
    {
        // Custom logic
        await _next(context);
    }
}
```

- Use in `Startup.cs`:

```
app.UseMiddleware<MyMiddleware>();
```

Routing

- **Attribute Routing**

- Define routes directly on actions:

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

- **Convention-Based Routing**

- Define in `Startup.cs`:

```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
```

MVC (Model-View-Controller)

- Controllers

- Handle requests and return responses.
- Example:

```
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```

- Views

- Razor syntax: `@model MyModel`
- Example:

```
@model MyModel
<h1>@Model.Title</h1>
```

- Models

- Data structures used in the application.
- Example:

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

Razor Syntax

- Basic Syntax

- `@`: Razor syntax prefix.
- `@Model`: Access model properties.
- `@if`, `@foreach`: Control structures.

- Example

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

Entity Framework Core

- **DbContext**

- Represents a session with the database.
- Example:

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

- **Migrations**

- Create and apply database migrations.
- Commands:

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

Authentication and Authorization

- **Authentication**

- Use `Microsoft.AspNetCore.Authentication` packages.
- Example:

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

- **Authorization**

- Use `Authorize` attribute:

```
[Authorize]
public class MyController : Controller
{
    // Actions
}
```

Logging

- **Logging Providers**

- Configure in `Startup.cs`:

```
public void Configure(IApplicationBuilder app, ILoggerFactory
loggerFactory)
{
    loggerFactory.AddConsole();
}
```

- **Logging Levels**

- `Trace`, `Debug`, `Information`, `Warning`, `Error`, `Critical`.
- Example:

```
_logger.LogInformation("Information message");
```

Testing

- **Unit Testing**

- Use `xUnit`, `NUnit`, or `MSTest`.
- Example:

```
[Fact]
public void TestMethod()
{
    // Arrange
    var service = new MyService();

    // Act
    var result = service.GetData();

    // Assert
```

```
        Assert.Equal("ExpectedData", result);
    }
```

- **Integration Testing**

- Use `Microsoft.AspNetCore.TestHost`.
- Example:

```
var server = new TestServer(new WebHostBuilder()
    .UseStartup<Startup>());
var client = server.CreateClient();
```

Deployment

- **Publish Command**

- Publish to a directory:

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

- **Docker Support**

- Create a Dockerfile:

```
FROM mcr.microsoft.com/dotnet/aspnet:5.0
COPY ./publish /app
WORKDIR /app
ENTRYPOINT ["dotnet", "MyApp.dll"]
```

Best Practices

- **Separation of Concerns**

- Keep controllers lean; move business logic to services.

- **Error Handling**

- Use global exception handlers.
- Example:

```
app.UseExceptionHandler("/Home/Error");
```

- **Security**

- Use HTTPS.
- Validate inputs.
- Implement proper authentication and authorization.

Tools and Extensions

- **Visual Studio**

- Integrated development environment for ASP.NET Core.

- **Visual Studio Code**

- Lightweight editor with extensions for ASP.NET Core.

- **NuGet**

- Package manager for .NET.
- Example:

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

Common Commands

- **Project Creation**

- Create a new project:

```
dotnet new webapp -n MyApp
```

- **Running the Application**

- Run the application:

```
dotnet run
```

- **Adding Packages**

- Add a NuGet package:

```
dotnet add package MyPackage
```

Conclusion

- **ASP.NET Core** is a powerful framework for building modern web applications.
- **Key Concepts**: MVC, Dependency Injection, Middleware, Entity Framework Core.
- **Best Practices**: Separation of Concerns, Error Handling, Security.
- **Tools**: Visual Studio, Visual Studio Code, NuGet.

By Ahmed Baheeg Khorshid

ver 1.0