ASP.NET Core Best Practices

Shailendra Chauhan

Microsoft MVP, Technical Consultant & Corporate Trainer

DotNetTricks

Agenda

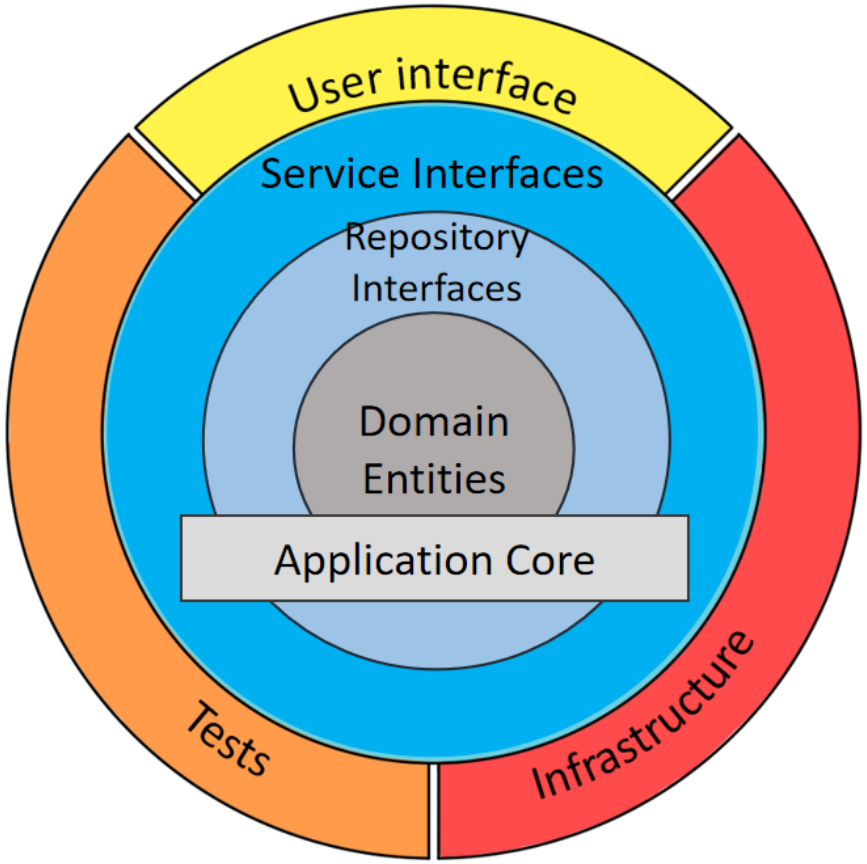
- Best Practices for Code
  - Architecture and Project Organization
  - Environment Based Settings
  - Handle Errors Globally
  - Routing For REST APIs
  - Error Logging
- Best Practices for Security
  - Security Using JWT and ASP.NET Core Identity
  - Prevent Cross-Site Scripting
  - Prevent SQL Injection Attack
  - Prevent Cross-Site Request Forgery (CSRF)
  - Hide the Version Information
  - Enforce SSL and use HSTS
  - Secure Your Cookies
- Best Practices for Performance
  - Avoid blocking calls
  - Bundle Your JS and CSS Files
  - Use CDN to Serve Content
  - Use Caching
  - Response Compression
  - HTML Minification

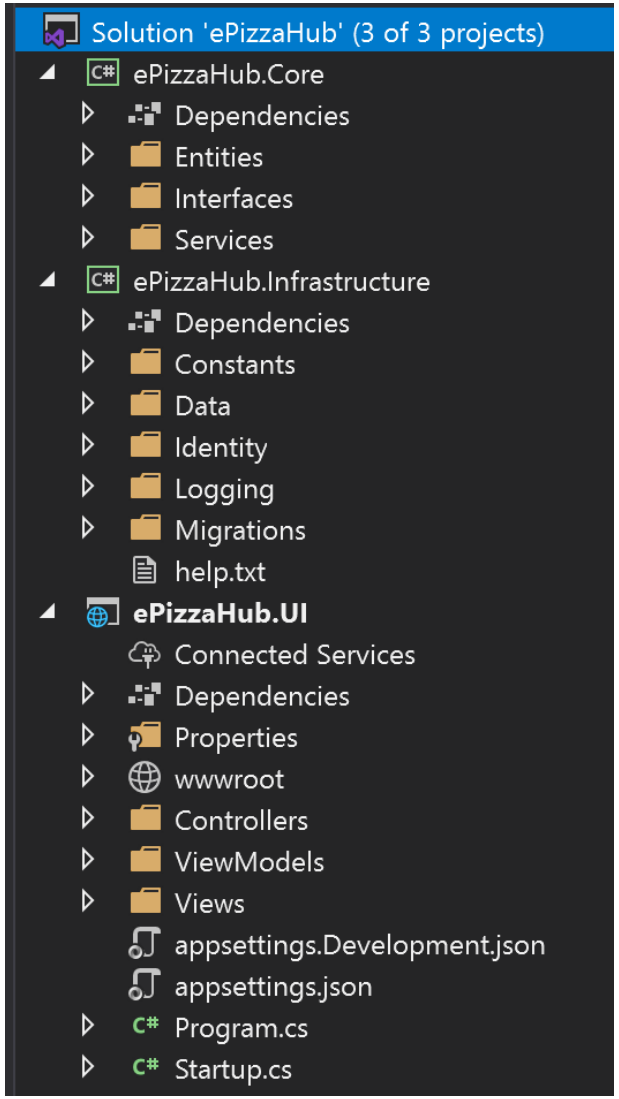
Best Practices For:

ASP.NET Core Code

DotNetTricks

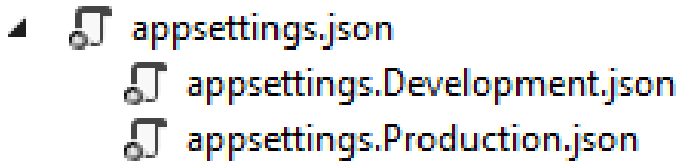
Project Organization





DotNetTricks

Environment Based Settings




```
public class Startup
{
    private readonly IHostingEnvironment _env;
    public Startup(IHostingEnvironment env) {
        _env = env;
    }
    public void ConfigureServices(IServiceCollection services)
    {
        if (_env.IsDevelopment()) {
            // Development environment code
        }
        else {
            // Code for all other environments
        }
    }
    public void Configure(IApplicationBuilder app)
    {
        if (_env.IsDevelopment()) {
            // Development environment code
        }
        else {
            // Code for all other environments
        }
    }
}
```

Handle Errors Globally

```
public class CustomExceptionMiddleware
{
    public async Task Invoke(HttpContext httpContext)
    {
        try
        {
            await _next(httpContext);
        }
        catch (Exception ex)
        {
            _logger.LogError("Unhandled exception ...", ex);
            await HandleExceptionAsync(httpContext, ex);
        }
    }
}
```

```
public static IApplicationBuilder UseCustomExceptionMiddleware(this IApplic
ationBuilder builder)
{
    return builder.UseMiddleware<CustomExceptionMiddleware>();
}
```

DotNetTricks

Handle Errors Globally Contd..

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

Routing For REST APIs

```
[Route("api/product")]
public class ProductController : Controller
{
    [HttpGet]
    public IActionResult GetAllProducts()
    {
    }

    [HttpGet("{id}")]
    public IActionResult GetProductById(Guid id)
    {
    }
}
```

Error Logging

```
public class AccountController: Controller
{
    private readonly ILogger _logger;

    public TestController(ILogger<AccountController> logger)
    {
        _logger = logger;
    }

    public IActionResult Login()
    {
        _logger.LogInfo("Here is info message from the account controller.");
        // TO DO:
        return View();
    }
}
```

Best Practices For:

ASP.NET Core Security

DotNetTricks

Security Using JWT

```
public void ConfigureServices(IServiceCollection services)
{
     services.AddAuthentication(opt => {
        opt.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        opt.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(options => {
          options.TokenValidationParameters = new TokenValidationParameters {
             //Configuration in here
          };
       });
}
```

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseAuthentication();
    app.UseAuthorization();
}
```

Security Using ASP.NET Core Identity

Prevent Cross-Site Scripting

- Script injection can be carried out in the following ways:
  - Form Inputs
  - URL Query Strings
  - HTTP Headers
- Cross-site scripting attacks prevention can be done by:
  - HTML Encoding: MVC Razor engine automatically encodes all inputs so that the script part provided in any field will never be executed.
  - URL Encoding: Encode the query parameter input in the URL.

```
string encodedValue = System.Net.WebUtility.UrlEncode("raw-string-text");
string decodedValue = System.Net.WebUtility.UrlDecode(encodedValue);
```

  - Regular Expression

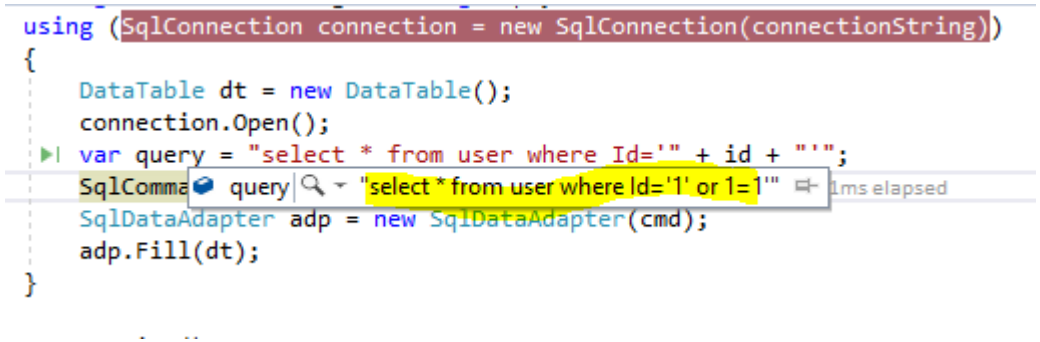
Regular Expression

```
public class UserModel
{
    // Allow up to 40 uppercase and lowercase
    // characters. Use custom error.
    [RegularExpression(@"^[a-zA-Z''-'\s]{1,40}$",
         ErrorMessage = "Characters are not allowed.")]
    public object FirstName;

    // Allow up to 40 uppercase and lowercase
    // characters. Use standard error.
    [RegularExpression(@"^[a-zA-Z''-'\s]{1,40}$")]
    public object LastName;
}
```

Prevent SQL Injection Attack

- Validate inputs: Server side
- Use parameterized queries
- Use stored procedures
- Use Entity Framework or any other ORM



Prevent Cross-Site Request Forgery (CSRF)

- Use AntiForgeryToken and Validate it Action Level

```
<form asp-controller="Account" asp-action="Register" asp-antiforgery="true">
   <div>Username : <input type="text" asp-for="Username"/></div>
   <div>Username : <input type="password" asp-for="Password"/></div>
   <div><input type="submit" value="Login"/></div>
</form>
```

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Login(UserLogin model)
{

}
```

Hide the Version Information

- A response header contains the following information:
  - Server: Microsoft-IIS/10.0
  - x-powered-by: ASP.NET
  - x-sourcefiles:

```
<httpProtocol>
  <customHeaders>
    <remove name="X-Powered-By" />
  </customHeaders>
</httpProtocol>
```

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                  webBuilder.ConfigureKestrel(serverOptions =>
                  {
                    serverOptions.AddServerHeader = false;
                  });
                    webBuilder.UseStartup<Startup>();
                });
```

Enforce SSL and use HSTS

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseHsts();
    }
    app.UseHttpsRedirection();
    app.UseRouting();
    app.UseEndpoints(endpoints =>
    {

    });
}
```

Secure Your Cookies

```
public void ConfigureServices(IServiceCollection services)
{
  services.Configure<CookiePolicyOptions>(options =>
  {
    options.MinimumSameSitePolicy = SameSiteMode.Strict;
    options.HttpOnly = HttpOnlyPolicy.Always;
    options.Secure = Env.IsDevelopment() ? CookieSecurePolicy.None : CookieSecurePolicy.Always;
  });
}

public void Configure(IApplicationBuilder app, IHostingEnvironment hostingEnvironment)
{
    app.UseCookiePolicy();
}
```

Best Practices For:

ASP.NET Core Performance

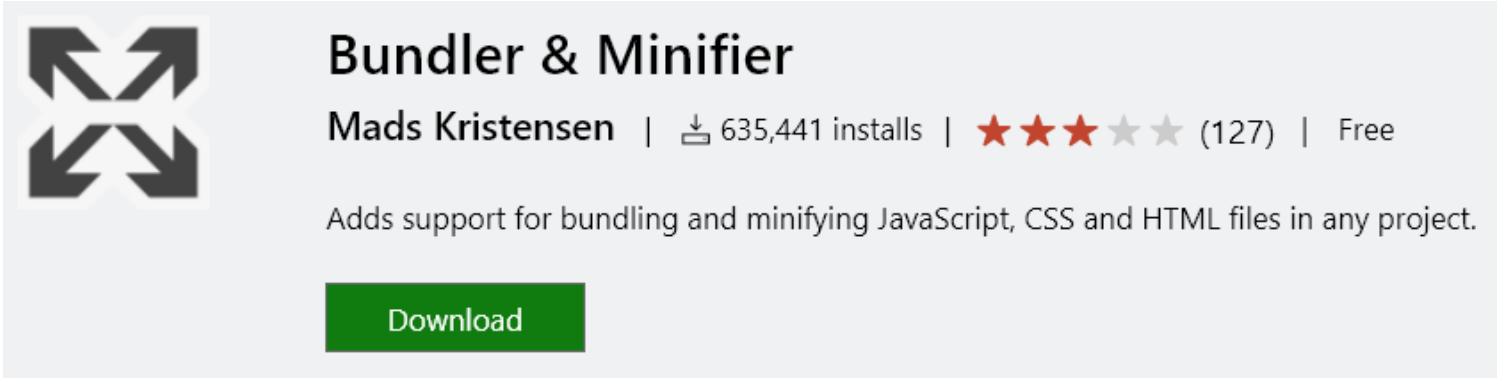
DotNetTricks

Avoid blocking calls

```
public async Task<List> GetUsersAsync()
{
    using (var context = new AppDbContext())
    {
        return await context.Users.ToListAsync();
    }
}

public static async Task AddUserAsync(User user)
{
    using (var context = new AppDbContext())
    {
        context.Users.Add(user);
        await context.SaveChangesAsync();
    }
}
```

Bundle Your JS and CSS Files

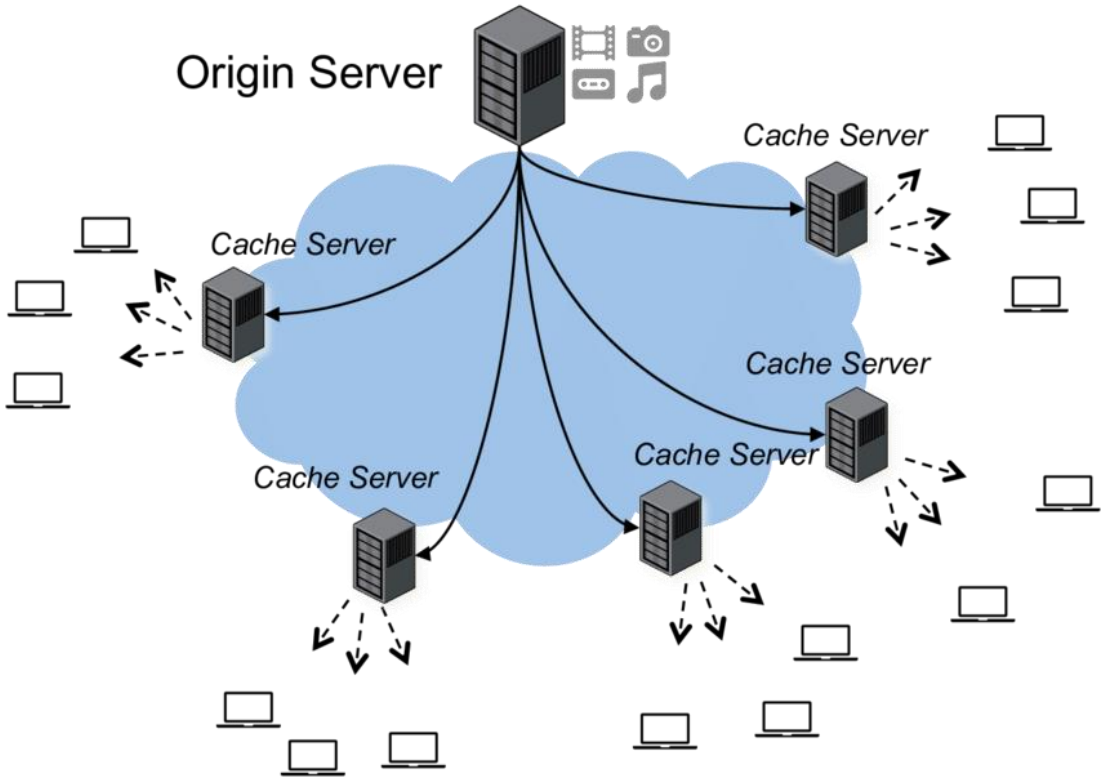



```
[{
    "outputFileName": "wwwroot/css/site.bundle.css",
    "inputFiles": [
      "wwwroot/css/bootstrap.css",
      "wwwroot/css/site.css"]
  },
  {
    "outputFileName": "wwwroot/js/jquery.bundle.js",
    "inputFiles": [
      "wwwroot/js/jquery.js",
      "wwwroot/js/bootstrap.js",
      "wwwroot/js/jquery.cookie.js"]
  }]
```

DotNetTricks

Use CDN to Serve Content

- Use a content delivery network (CDN) to load static files such as images, JS, CSS, etc.



Use Caching

- In-Memory Cache
- Response Cache
- Distributed Cache

```
public class HomeController : Controller
{
   [ResponseCache(VaryByHeader = "User-Agent", Duration = 30)]
   public IActionResult Index(){

   }
}
```

```
public class HomeController : Controller {
    private IMemoryCache _cache;
    public HomeController(IMemoryCache memoryCache) {
        _cache = memoryCache;
    }
    public IActionResult Index() {
       string key = "mykey";
       var cacheEntry = cache.GetOrCreate(key, entry => {
                 entry.SlidingExpiration = TimeSpan.FromMinutes(15);
                  return  GetData();
             });
         return View(cacheEntry);
    }
}
```

DotNetTricks

Response Compression

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddResponseCompression(options =>
    {
        options.Providers.Add<GzipCompressionProvider>();
    });
}
```

HTML Minification

WebMarkupMin.AspNetCore3

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddWebMarkupMin().AddHtmlMinification(options =>
        {
            options.MinificationSettings.RemoveRedundantAttributes = true;
            options.MinificationSettings.MinifyInlineJsCode = true;
            options.MinificationSettings.MinifyInlineCssCode = true;
        })
        .AddHttpCompression();
}
```