.NET Core SDK offerings

* CLI
  + Dotnet.exe
    - The runtime to execute the .NET Core apps as “In-Process” apps.
      * Windows
      * Linux
      * Mac OSX
    - Hosts and Executes
      * Console Apps
      * Class-Libraries
      * ASP.NET Core Apps
  + Donet new <Application Types>
    - Console
    - Class-Lib
    - Empty App
    - MVC App
    - Web API App
    - Test App
  + Dotnet Switches
    - New
    - Build
    - Run
    - Ef 🡪 EntitytFramework
      * Create Migration
      * Update Database

.NET Core App Development IDEs

1. Visual Studio 2017 with 15.5+ Update
   1. .NET Core 2.x SDK installed
2. Visual Studio Code (VSCode)
   1. Node.js
   2. .NET Core 2.x SDK
   3. OS: Linux and MAC OSX
3. Visual Studio for MAC
   1. MAC OSX

ASP.NET Core

1. ASP.NET Core Object Model
   1. Core Objects of ASP.NET Core
      1. Views
         1. Razor Views aka cshtml
      2. Controllers
         1. MVC Controllers base class as **Controller**
         2. ApiController base class as **ControllerBase**
            1. **ApiControllerAttribute class in ASP.NET Core 2.2**
      3. Tag-Helpers
         1. Starts from **asp-**
         2. Smart Attributes for HTML 5 elements. These are used for
            1. Model-Binding

asp-for

<input type=”text” asp-for=”EmpNo”/>

* + - * 1. Controller Binding

asp-controller

* + - * 1. Action Binding

asp-action

* + - * 1. Collection Iteration

asp-items

* + - * 1. Routing

asp-route

* + 1. Validation Annotations
       1. System.ComponentModel.DataAnnotations.dll
    2. Caching
       1. In-Memory Cache
    3. Identity Management
       1. Authorize
  1. Application Objects, Objects used for developing application
     1. Entities / POCO
     2. Repositories aka Services
     3. Custom Validators
     4. Domain Classes
     5. Custom tag-Helpers
     6. Custom Identity Model
  2. Infrastructure, generally provided by or arranged on Cloud
     1. Entity-Framework Providers
        1. Sql Azure
     2. Identity Providers
        1. Azure AD
        2. Third-Party Identity Providers
           1. Google
           2. Facebook
           3. Twitter
           4. Microsoft-Live
     3. Two-Factor Auth. Providers
        1. SendGrid
        2. Twillo
     4. Caching Services
        1. RedisCache
     5. Communication
        1. Azure ServiceBus

1. EF-Core Migrations
   1. Enabling Migrations
      1. Dotnet ef migrations add InitialCreate - -context <Full Qualify path of DbContext class> - -output-dir <Directory-Path for storing Migrations>
   2. Update Database
      1. Dotbet ef database update -c <Full Qualify path of DbContext class>

Startup.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Threading.Tasks;  using Microsoft.AspNetCore.Builder;  using Microsoft.AspNetCore.Identity;  using Microsoft.AspNetCore.Identity.UI;  using Microsoft.AspNetCore.Hosting;  using Microsoft.AspNetCore.Http;  using Microsoft.AspNetCore.HttpsPolicy;  using Microsoft.AspNetCore.Mvc;  using Microsoft.EntityFrameworkCore;  using Core\_WebApp.Data;  using Microsoft.Extensions.Configuration;  using Microsoft.Extensions.DependencyInjection;  using Core\_WebApp.Models;  namespace Core\_WebApp  {  public class Startup  {  public Startup(IConfiguration configuration)  {  Configuration = configuration;  }  public IConfiguration Configuration { get; }  // This method gets called by the runtime. Use this method to add services to the container.  public void ConfigureServices(IServiceCollection services)  {  services.Configure<CookiePolicyOptions>(options =>  {  // This lambda determines whether user consent for non-essential cookies is needed for a given request.  options.CheckConsentNeeded = context => true;  options.MinimumSameSitePolicy = SameSiteMode.None;  });    // This code will create an instance of The DbContext class  // in default DI container  services.AddDbContext<ApplicationDbContext>(options =>  options.UseSqlServer(  Configuration.GetConnectionString("DefaultConnection")));  // This code will create an instance of AppDbContext by registering  // it in Default DI Container. This instance will be ready for  // constructor injection  services.AddDbContext<AppDbContext>(options =>  options.UseSqlServer(  Configuration.GetConnectionString("AppDbContextConnection")));  services.AddDefaultIdentity<IdentityUser>()  .AddDefaultUI(UIFramework.Bootstrap4)  .AddEntityFrameworkStores<ApplicationDbContext>();  // load all dependant objects for MVC application  // 1. filters  // 2. webapi response formats  // 3. MVC Execution process  services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_2);  }  // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.  // IApplicationBuilder --> object used for defining and configuring application middlewares  // Middleware --> Value added objects in Request Pipeline for  // 1. Security  // 2. Security Policies  // 3. CORS  // 4. Custom Middleware management e.g. Global Error Handling/Logging etc.  // 5. Routing for MVC applications  // IHostingEnvironment --> Used to detect the Hosting Env. e.g. IIS, apache etch  // and load the runtime modules accrodingly  public void Configure(IApplicationBuilder app, IHostingEnvironment env)  {  if (env.IsDevelopment())  {  app.UseDeveloperExceptionPage();  app.UseDatabaseErrorPage();  }  else  {  app.UseExceptionHandler("/Home/Error");  // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.  app.UseHsts();  }  app.UseHttpsRedirection();  app.UseStaticFiles();  app.UseCookiePolicy();  app.UseAuthentication();  app.UseMvc(routes =>  {  routes.MapRoute(  name: "default",  template: "{controller=Home}/{action=Index}/{id?}");  });  }  }  } |

ASP.NET Core Programming

1. Model Approaches for EF Core
   1. Code-First with DbContext Migrations
   2. Database First
      1. Scaffold-Database <ConnectString> --output Model/
2. Controllers
   1. Object that manage
      1. View aka Resources
      2. Data aka API
   2. Controller, base class for MVC Controllers
   3. ControllerBase, the base class for WEB API Controllers
      1. The **ApiControllerAttribute** class, the class that provides WEB API Request Processing
3. Request Object Model for MVC and WEB API in .NET Core
   1. ActionContext, the base class for MVC and API Controllers
      1. Executes the Request based on Request Type (?)
         1. GET (Default)
         2. POST
         3. PUT
         4. DELETE
   2. IActionResult
      1. Common Contract for View Responses and API (Http Code and Data) responses
      2. E.g.
         1. ViewResult() is an instance class, this can be explicitely instantiated.
4. Views
   1. Razor Views with Base class as RazorView<TModel>
      1. TModel is the Model class passed to view for Model-binding
   2. Templates
      1. List, The TModel will be IEnumerable
      2. Create, The TModel will be empty Object
      3. Edit, The TModel will be object with values to be edited
      4. Delete, The TModel will be object which is to be deleted
      5. Details, The TModel will be read-only object
      6. Empty (Without Model), free-hand HTML view
5. Validations
   1. Data Annotations
      1. ModelState.IsValid, to validate all Post requests for the Model
   2. Process Based Validations
      1. Exception Handling Local to Action Method
         1. Try..catch block
            1. ErrorViewModel class in the Models folder
      2. Exception Filters for IServiceCollection for MVC Application
         1. Configure the ASP.NET Core Services in Http Pipeline for loading Exception Filters only for MVC Context
            1. Services.AddMvc();
            2. IMvcBuilder is the Contract for MVC Context for MVC object model for request processing

Action Filters for MVC request

This uses **MvcOptions** class for defining Additional Objects to be loaded for MVC Request processing

1. ASP.NET Core Cross-Controller State Management
   1. TempData
   2. Session
      1. ISessionProvider
         1. Serialized Data in state of standard preemptive types
            1. Int
            2. String
         2. HttpContext.Session.Set(“Key”, byte[]);
         3. HttpContext.Seesion.TryGet(“Key”,out value);
      2. Enable Session from the Middleware

ASP.NET Core for LOB Apps

1. Front-End Integration of ASP.NET Core apps
   1. Angular
   2. React
   3. React+Redux
2. WEB API
   1. New generation services for Customer Focused Apps
      1. Http Methods
3. Extensibility
   1. Custom Middleware Management
4. Security
   1. User and Role Management
   2. Role Policies
   3. JWT, to be used only in case of Front-End client apps for WEB API in .NET Core
   4. Security classes
      1. IdentityUser
         1. To manage application users
      2. UserManager<IdentityUser>
         1. Manage Users for Storing in DB as well as reading from DB
      3. IdentityRole
         1. To manage roles
      4. RoleManager<IdentityRole>
         1. Create and Read Roles to and from DB
   5. To custom Identity in ASP.Net Core add the Scaffolding for **Identity**
      1. This is a Web Form Application provided as **Razor Library** in ASP.NET Core 2.2
5. Razor View Library for Identity in ASP.NET Core 2.2
   1. In-build WebForm Apps for Security
   2. Views for
      1. Register
      2. Login
      3. ….