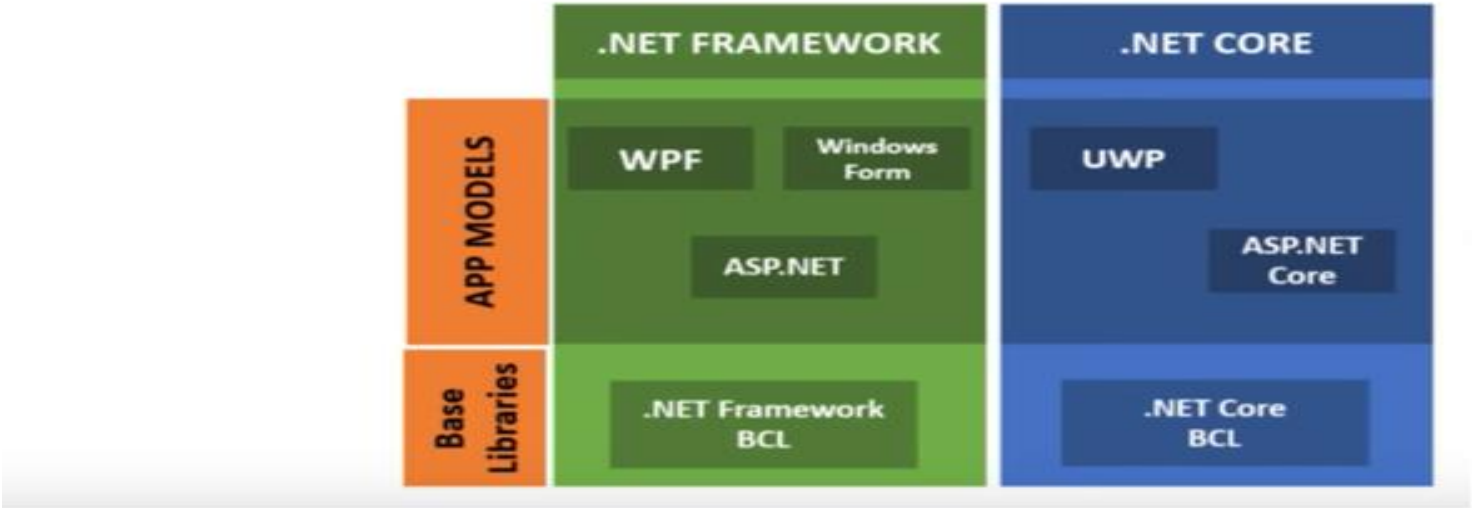
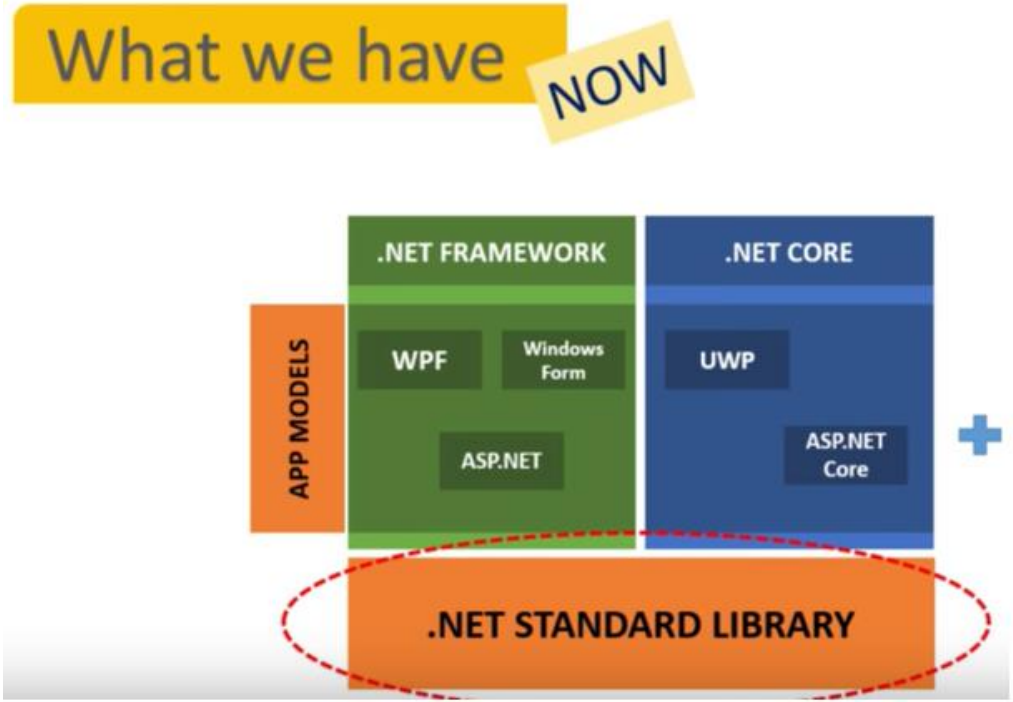
Web Application Development using ASP.NET Core

Contents

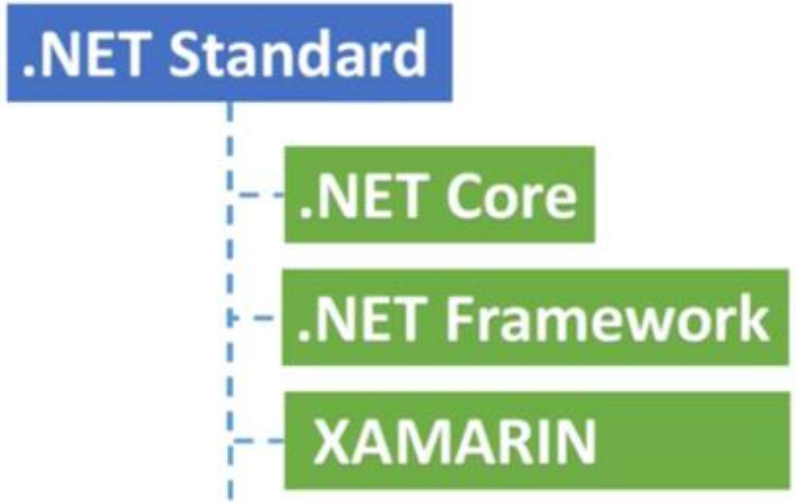
- Understand basics of .NET Core and its architecture
- ASP.NET Core Features
- ASP.NET Core Project Structure
- Working with Middlewears
- Hosting in ASP.NET Core Applications
- ASP.NET Core MVC and Entity framework Core
- Logging and Exception handling in ASP.NET Core
- Authentication and Authorization

What we had Earlier




What we have
NOW
.NET FRAMEWORK
.NET CORE
APP MODELS
WPF
Windows Form
UWP
ASP.NET
ASP.NET Core
.NET STANDARD LIBRARY

What is .NET STANDARD



.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0	2.1
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0	3.0
.NET Framework [1]	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1 [2]	4.6.1 [2]	4.6.1 [2]	N/A[3]
Mono	4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4	6.4
Xamarin.iOS	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.14	12.16
Xamarin.Mac	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.8	5.16
Xamarin.Android	7.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0	10.0
Universal Windows Platform	10.0	10.0	10.0	10.0	10.0	10.0.16299	10.0.16299	10.0.16299	TBD
Unity	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	2018.1	TBD

.NET Core

Overview of .NET Core

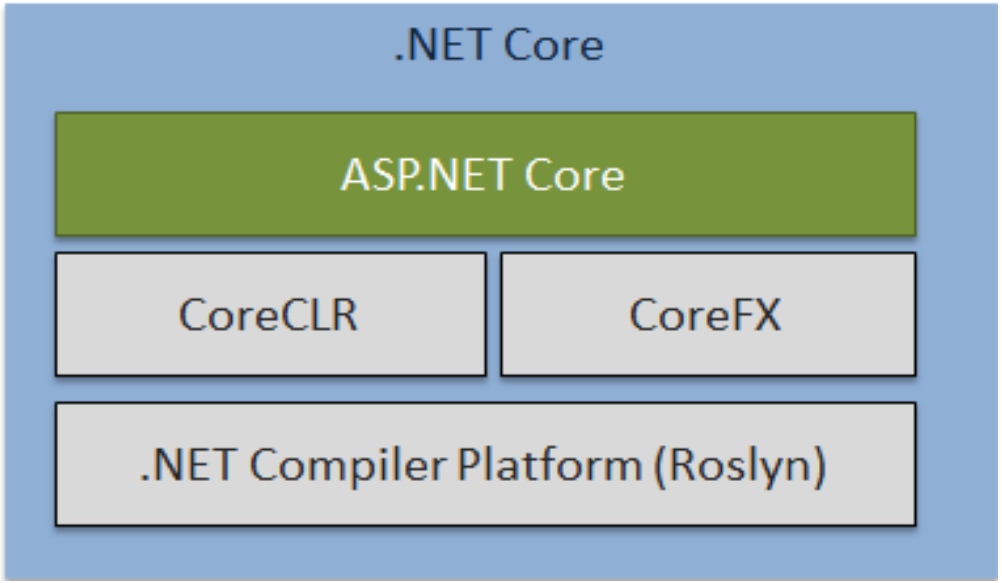
- .NET Core is the latest general purpose development platform maintained by Microsoft. It works across different platforms and has been redesigned in a way that makes .NET fast, flexible and modern.
- .NET Core applications now build Android, iOS, Linux, Mac, and Windows applications with .NET, all in Open Source.

.NET Core Characteristics

- Open Source
- Cross Platform
  - Application implemented in .NET Core can be run and its code can be reused regardless of your platform targets.
  - Currently it supports
    - Windows
    - Linux
    - MacOS
    - Android
    - iOS
- Flexible Deployment
- Modular

.NET Core

.NET Core Components



ASP.NET Core

ASP.NET Core

Overview

- ASP.NET Core is a free, open-source and cloud optimized web framework which can run on Windows, Linux, or Mac.
- ASP.NET Core is a modular framework distributed as NuGet packages. This allows us to include packages that are required in our application.
- ASP.NET Core applications run on both, .NET Core and traditional .NET framework (.NET Framework 4.x).

ASP.NET Core

Environment Setup

- To develop ASP.NET Core application, the following must be installed in your system:
  - .NET Core SDK
  - Integrated Development Environment (IDE)
- ASP.NET Core is a part of .NET Core SDK, so you don't need to install it separately.
- To download .NET Core SDK use below url and download it for the appropriate OS.

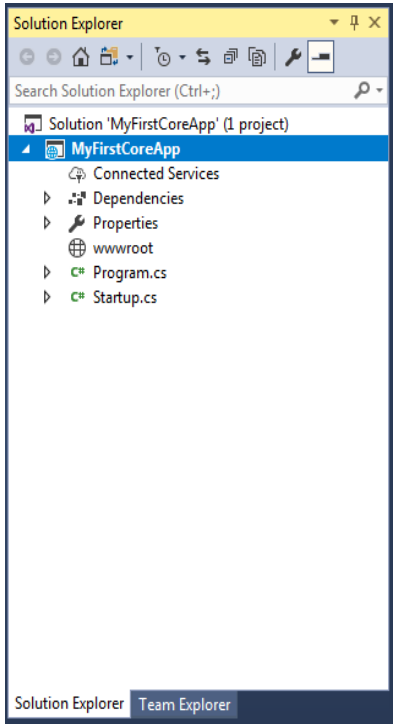
  https://www.microsoft.com/net/core
- To download IDE, Visual Studio 2017 from below url.

  https://www.visualstudio.com/downloads/

If you do not use Visual Studio for .NET core application development for some reason and want to use different IDE then you can use command-line interface to create, compile, build, restore and run .NET Core application.

ASP.NET Core

Project Structure



By default, the **wwwroot** folder in the ASP.NET Core project is treated as a web root folder. Static files can be stored in any folder under the web root and accessed with a relative path to that root.

Only those files that are in the web root - wwwroot folder can be served over an http request. All other files are blocked and cannot be served by default.

You can rename wwwroot folder to any other name as per your choice and set it as a web root while preparing hosting environment in the program.cs.

For example,wwwroot folder has been renamed to Content folder.
Call UseWebRoot() method to configure Content folder as a web root folder in the Main() method of Program class

ASP.NET Core web application is actually a console project which starts executing from the entry point public static void Main() in Program class where we can create a host for the web application.

ASP.NET Core-Project Structure

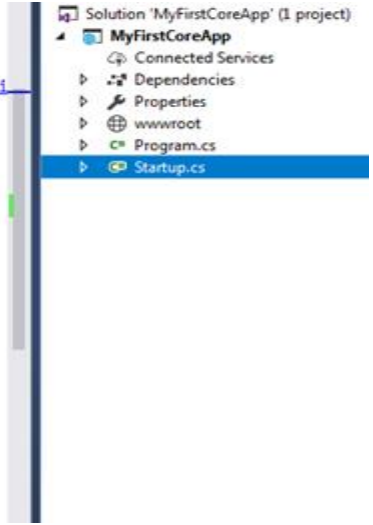
StartUp Class

```
public class Startup
{
    // This method gets called by the runtime. Use this method to add services to the container.
    // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?Li
    public void ConfigureServices(IServiceCollection services)
    {
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        app.Run(async (context) =>
        {
            await context.Response.WriteAsync("Hello World!");
        });
    }
}
```

Register Dependent Types (Services) with IoC Container here

Configure HTTP request pipeline (Middleware) here



The Configure method includes three parameters IApplicationBuilder to define Http Request Pipeline, IHostingEnvironment, and ILoggerFactory by default. These services are framework services injected by built-in IoC container.
At run time, the ConfigureServices method is called before the Configure method. This is so that you can register your custom service with the IoC container which you may use in the Configure method.

ASP.NET Core

Command Line interface

Basic Commands	Description
new	Creates a new project, configuration file, or solution based on the specified template.
restore	Restores the dependencies and tools of a project.
build	Builds a project and all of its dependencies.
Run	Runs source code without any explicit compile or launch commands.
publish	Packs the application and its dependencies into a folder for deployment to a hosting system.
test	Executes unit tests.
vtest	Runs tests from the specified files.
pack	Packs the code into a NuGet package.
clean	Cleans the output of a project.
sln	Modifies a .NET Core solution file.
help	Display help on the specified command
store	Stores the specified assemblies in the runtime package store.

- The .NET Core command-line interface (CLI) is a new cross-platform tool for creating, restoring packages, building, running and publishing .NET applications.
- The following is a command structure.

  dotnet <command> <argument> <option>

  All the commands start with driver named dotnet.

Middlewears

ASP.NET Core

Middlewears

- A middleware is nothing but a component (class) which is executed on every request in ASP.NET Core application.
  - Built-In Middlewear
    - UseDeveloperExceptionPage(Shows DeveloperException Page in case development enviornment)
    - UseDefaultFiles()(Changes request path to the default page to run)
    - UseStaticFiles()(To Serve static files from wwwroot folder)
    - UseDefaultFiles() middlewear will need UseStaticFiles() middlewear to process the request further to process default file.
    - In MVC Core application, we might want some other controller to handle all exceptions and display custom user friendly error messages. The UseExceptionHandler extension method allows us to configure custom error handling route. This is useful when an application runs under production environment.
      - For example
        - app.UseExceptionHandler("/Home/Error");
    - Custom Middlewear
- Configure StaticFiles Middleware
  - app.UseStaticFiles(new StaticFileOptions() { FileProvider = new PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(), "Content")), RequestPath = new PathString("/Admin") });
- To download IDE, Visual Studio 2017 from below url.

  https://www.visualstudio.com/downloads/

If you do not use Visual Studio for .NET core application development for some reason and want to use different IDE then you can use command-line interface to create, compile, build, restore and run .NET Core application.

ASP.NET Core

Configure StaticFile Middlewear

- Configure StaticFiles Middleware
  - app.UseStaticFiles(new StaticFileOptions() { FileProvider = new PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(), "Content")), RequestPath = new PathString("/Admin") });
- Built in logging providers in ASP.NET Core.
  - 1. Console--logs msgs on console
  - 2. Debug
  - 3. EventSource-
  - 4. EventLog--windows os and want to log msgs to windows event log
  - 5. TraceSource
  - 6. AzureAppServicesFile--logs msgs to AzureAppServicesFile
  - 7. AzureAppServicesBlob
  - 8.ApplicationInsights

Third Party logging providers also helps in logging exceptions.

For example Nlog and Serilog providers helps to log the exceptions in a file.

ASP.NET Core MVC and Entity Framework Core

ASP.NET Core

ViewComponent, ViewImports, and TagHelpers

- ViewComponent
  - ViewComponent is very much similar to partial view.
  - It is class inherited from ViewComponent
  - In most of the cases it returns View
  - It does not have any UI
  - It can be invoked from Controller using ViewComponent() method
  - It can be invoked from a view using Taghelper or by calling InvokeAsync method.
- _ViewImports.cshtml
  - The ViewImports file is a place where we can write code and place common directives to pull in namespaces that our views need.
  - If there are namespaces that we commonly use in our views, we can have **using directives** appear once in our **ViewImports** file instead of having **using directives** in every view or typing out the full namespace of a class.
- TagHelpers
  - Tag Helpers enable server-side code to participate in creating and rendering HTML elements in Razor files. Tag helpers are a new feature and similar to HTML helpers, which help us render HTML.

ASP.NET Core

Adding Session State

```
public class Startup {
      public void Configure(IApplicationBuilder app)
      {
         app.UseSession();
         app.UseMvc();
         app.Run(context => {
            return context.Response.WriteAsync("Hello Readers!");
         });
      }
      public void ConfigureServices(IServiceCollection services)
      {
         services.AddMvc();
         services.AddSession(options => {
            options.IdleTimeout = TimeSpan.FromMinutes(30);
         });
      }
   }
```

Dependency Injection

- ASP.NET Core framework contains simple IoC container which does not have as many features as other third party IoC containers.
- For more features such as auto-registration, scanning, interceptors, or decorators then you may replace built-in IoC container with a third party container.
- The built-in container is represented by IServiceProvider implementation that supports constructor injection by default. The types (classes) managed by built-in IoC container are called services.
- Registering Application Service
  - AddSingleton()
  - AddScoped()
  - AddTransient()
- Constructor Injection
  - Once we register a service, the IoC container automatically performs constructor injection if a service type is included as a parameter in a constructor.

ASP.NET Core

Entity Framework Core

Entity Framework Core is an ORM framework which has been rewritten over Entity Framework.
To work with EntityFrameworkCore we need to install below packages.

- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.EntityFrameworkCore.Relational
- Microsoft.EntityFrameworkCore

- While installing it need to follow the sequence as once we install Microsoft.EntityFrameworkCore.SqlServer

Package it installs other two automatically.

EntityFrameworkCore supports below 2 approaches:

1. Database First
2. Code First

To work with EntityFrameworkCore and database a class inherited from DbContext class and a property of DbSet type is required

ASP.NET Core

Entity Framework Core

Migration is EntityFrameworkCore feature which keeps the database schema and application model classes in sync.

Commands to work with migration

1. To create migration

   Add-Migration <migration name>

   2. TO apply/run migration

   update-database <migration name>

   To seed initial data in database need to override OnModelreating method.

```
protected override void OnModelCreating(ModelBuilder builder)
      {
         builder.Entity<Employee>().HasData
            (
            new Employee() { eid = 1, ename = "Swati", dept = "Training", location = "Pune" },
            new Employee() { eid = 1, ename = "Swati", dept = "Training", location = "Pune" }
            );
      }
```

Hosting in ASP.NET Core

Hosting in ASP.NET Core

- Whenever you create an ASP.NET Core application, by default it contains an internal server provided by a .NET Core that is called Kestrel. Due to this server, we can run ASP.NET Core apps on any platform like Windows, Mac or Linux.

- **What is the Kestrel Server?**

- *Kestrel is a cross-platform web server for ASP.NET Core. Kestrel is the webserver that's included by default in ASP.NET Core project templates.*
- Kestrel is based on the libuv library, the same library which is used by Node.

- Some features of Kestrel:
- It supports SSL
- lightweight
- cross-platform

Hosting Models in ASP.NET Core

- Out-Of Process Hosting
  - In Out-of-process hosting models, we can either use the Kestrel server directly as a user request facing server or we can deploy the app into IIS which will act as a proxy server and sends requests to the internal Kestrel server. In this type of hosting model we have two options:
    - **Using Kestrel**
      - So in this type Kestrel itself acts as edge server which directly server user requests. It means that we can only use the Kestrel server for our application.
    - **Using a Proxy Server**
      - Due to limitations of the Kestrel server, we can not use this in all the apps. In such cases, we have to use powerful servers like IIS, NGINX or Apache. So, in that case, this server acts as a reserve proxy server which redirects every request to the internal Kestrel sever where our app is running. Here, two servers are running. One is IIS and another is Kestrel.
-

Hosting Models in ASP.NET Core

- **In-process Hosting Model**
  - After the release of .NET Core 2.2, it introduced a new type of hosting which is called In-process hosting. In this type, only one server is used for hosting like IIS, Nginx or Linux. It means that the App is directly hosted inside of IIS. No Kestrel server is being used. IIS HTTP Server (IISHttpServer) is used instead of the Kestrel server to host apps in IIS directly. ASP.NET Core 3.1 onwards **In-process** hosting model is used as a default model whenever you create a new application using an existing template.

Logging and Exception Handling

Exception Handling

- UseDeveloperExceptionPage
  - Shows Developer exception page if the enviorment is set to Development
- UseExceptionHandler
  - In case to execute custom exception handler, use UseExceptionHandler middlewear

Logging In ASP.NET Core

- When you create the ASP.NET Core MVC web application in Visual Studio 2017 (or later), it automatically includes the NuGet package for Microsoft.Extensions.Logging and the following logging providers under the Microsoft.AspNetCore.App NuGet package. So, we don't have to install it manually.
  - Microsoft.Extensions.Logging.Console
  - Microsoft.Extensions.Logging.Debug
  - Microsoft.Extensions.Logging.EventSource
  - Microsoft.Extensions.Logging.TraceSource

- Third Party Logging providers also can be configured for various special features
  - Nlog
  - Serilog etc.

- Create Logs in the Controller
  - Ilogger
  - ILoggerFactory

Authentication and Authorization

ASP.NET Core

Authentication and Authorization

- Identity Services
- IdentityDbContext
- Add Identity services using AddIdentity() method
- Use SignInManager and UserManager Services for managing users and Sign in Sign Out functionality
- External Login Providers like Microsoft, Google,Facebook, Twitter etc.
- Define Authorization Policy
- Implement Authorization using Authorize filter at Action Method Level/Controller Level or apply it globally.
- Cookie Based Authorization

Thank You

For more information please contact:
T+ 33 1 98765432
F+ 33 1 88888888
M+ 33 6 44445678
firstname.lastname@atos.net

Atos | Syntel