Compilation and Execution of .NET applications:

→ Any code written in any .NET compliant languages when compiled converts into MSIL code in form of assembly through - CLS, CTS.

→ IL is the language that CLR can understand.

→ On execution, this IL is converted into binary code by CLRs just in time compiler (JIT) and these assemblies are loaded into memory.

→ Compilation can be done with Debug or Release configuration. The difference between these two is that in the debug configuration, only an assembly is generated without optimization. However, in release complete optimization is performed without debug symbols.

ASP.NET Core Architecture Overview

1. ASP.NET Core Web App: Controllers, Response Caching filters, ASP.NET Core Identity, Views, Other filters.
   1. Infrastructure Project: In Memory Data Cache, SMS service, Email Service, EF Core DbContext, Redis cache, Azure Service Accessor, Other Web API Clients.
   2. Application Core Project: Interfaces, Domain Events, Aggregates, POCO Entities, Application Exceptions, Specifications, Business Services, Value Objects
      1. Data Sources: SQL, S3 | Redis Cache
      2. Third Party Services: Github API, Send Grid API

Rendering HTML with Views:

In ASP.NET Core MVC, views are .cshtml files that use the C# programming language in Razor markup. Usually view files are grouped into folders named for each of the app's controllers. The folders are stored in Views folder at the root of the app.

The Home controller is represented by a Home folder inside the Views folder. The Home folder contains the views for the About.cshtml, Contact.cshtml, Index.cshtml etc. webpages. When a user requests one of these three webpages, controller actions on the Home controller determine which of the three views is used to build and return webpage to the user.

Razor Syntax

Razor is a view engine for MVC framework. Razor is a markup syntax that lets us embeded server-based code into web pages. Server based code can create dynamic web content on the fly, while a webpage is written to the browser. It has the power of traditional ASP.NET (.ASPX) markup, but it is easier to use, flexible and lightweight. Razor markup starts with the @ symbol within html document.

Ex: Necessary html and C# code in razor page to display all numbers from 1 to 100. All the even numbers must be displayed in green color and odd numbers in red color.

Steps to add authentication to apps and identity service configurations:

Step1: Install All Package from Nuget.
- EntityFrameworkCore
- EntityFrameworkCore.SqlServer
- " .Tools
- " .Design

Step2: Adding AppDbContext inside Model folder and inherits from IdentityDbContext class.

```
public class AppDbContext : IdentityDbContext{
public class AppDbContext(DbContextOptions<AppDbContext> options):
base(options){
}
protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);
}
}
```

Q. How to publish asp.net core app in azure cloud?

Ans:
Step1: Right click in project and select publish option.
Step2: Select Azure as target.
Step3: Click next and select specific target and we will find Azure App Services in right pane.
Step4: After clicking Next we will find App Service in left pane and click (+) sign to create new App Service instances in right pane.
Step5: After clicking (+) sign we will see new popup window to add new Service instance which has fields like Name, Subscription, Hosting Plan. Now just click in new hosting plan from (+) sign. In hosting plan section we will select size option to free and click OK. And then click create. Now app service instance is created in Azure.

[Hosting Models]

Q. How do you host and deploy ASP.NET core application?

Ans: There are two types of hosting models in ASP.NET Core:

1) Out-of-process Hosting Model: In this model we can either use Kestral 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 Kestral server. In this type of hosting model we have two options:
   i) Using Kestral: Kestral is a cross platform web server for ASP.NET core. Kestral is the web server that is included by default in ASP.NET Core project templates. Kestral itself acts as an edge server which directly serve user requests. It means that we can only use the Kestral server for our application.
   ii) Using a Proxy Server: Due to limitations of the Kestral server, we cannot use this in all the apps. In such cases, we have to use powerful servers like IIS, NGIX or Apache. So, in that case, this server acts as a reverse proxy server which redirects every request

SPA features

No Server roundtrip: SPA can easily reshape any part of UI to restore the full HTML page preventing server roundtrip.

Templating and routing on client side: Supported by JS based routers, SPA can monitor the users actual state and location within the whole navigation experience.

The compatibility to work offline: A SPA can work offline in the event of a lost internet connection, and conn... is restored when online.

Cross platform functionality: SPA work on different OS from any browser.

A reduced throughput: The connection with server is carried out in small fragments. Data is packed into smaller objects like JSON data format.

```
<html>
<body>
    @for (int i=0; i<100; i++){
    if (i%2 ==0){
        <h2 style="color:green">@i</h2>
    }
    else{
        <h2 style="color:red">@i</h2>
    }
</body>
</html>
```

URL Routing

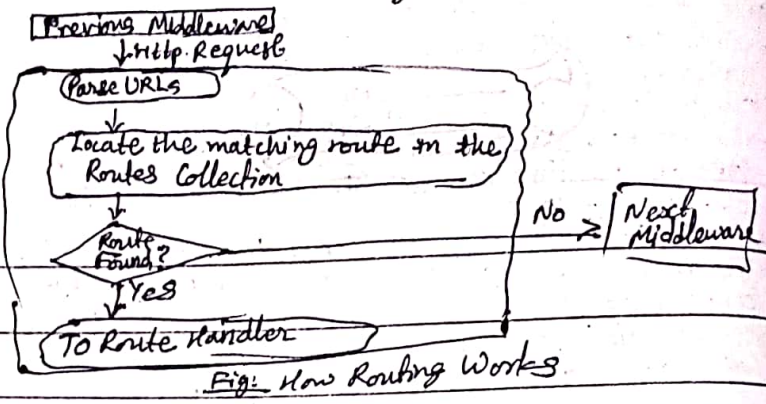

Fig: How Routing Works

The following program shows how view is displayed

1. HTML View Defined as /Views/Home/Index in file Index.cshtml.

```
<h1>This is Index Page</h1>
```

2. Action Method Index() defined at Controller /Home in file Controller.cs

```
using Microsoft.AspNetCore.Mvc;
namespace MyMvcApplication.Controllers{
    public class HomeController:Controller{
        public IActionResult Index(){
            return View();
        }
    }
}
```

Step3: Add Line of code inside Startup.cs inside ConfigureServices Method.

```
services.AddDbContextPool<AppDbContext>(options=>options.
    UseSqlServer("Database connection string"));
services.AddIdentity<IdentityUser,IdentityRole>().AddEntity
    FrameworkStores<AppDbContext>();
```

Step4: Add below line code inside Startup.cs inside Configure Method.

```
app.UseAuthentication();
```

Step5: Now, Go To Tools > Nuget Package Manager > Package Manager Console.

Type: Add-Migration AddingIdentity.

Step6: Type: Update Database.

Step6: Now we will see App Service Instance is created and select newly created instance and click Next.

Step7: We will redirect to API Management tab in right pane, this option for API management we can skip by selecting checkbox skip this step and click finish.

to the internal Kestral server where our app is running. Here two servers are running. One is IIS and another is Kestral.

2). In-process Hosting Model: 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 Kestral server is being used. IIS HTTP Server is used instead of the Kestral server to host apps in IIS directly.

#Steps to Deploy ASP.NET Core to IIS:

Step1: Publish to a File folder. Publish to folder with Visual Studio.

Step2: copy Files to Preferred IIS location.

Step3: Create application in IIS.

Step4: Load App!