NAME:- Shyed Shahriar Housaini (1)
ID:- MCE 07905536
Subject: Web Computing & Mining
Subject code: MCSE541

Ans. to the q. no- 1(b)

Basic Differences to display data on browser-without view (middleware only), with view (ViewBag etc); With view (@model)

Without view:- Middleware in ASP.NET Core controls how our application responds to HTTP requests. Middleware has access to both incoming & outgoing ~~requests.~~ responses and simply pass the request to the next piece of middleware in the pipeline.

(P.T.O)

Middleware may perform some logic (2)
then pass to next, may terminate or
short-circuit and they are executed
in order they are added to the pipeline.
They use root directory & endpoint
routing or support Razor pager.
example:

```
app.UseEndpoints(endpoints => {
    endpoint.MapRazorPages(); });
```

or direct Browsing

```
app.UseDirectoryBrowser(new DirectoryBrowser
        Options { FileProvider = new PhysicalFileBro
vider(Path.Combine(Directory.GetCurrentDirectory(),
"WWWroot", "images")),
        RequestPath = "/images" });
```

Or they use app.Run(), app.Use(), app.Map(). (P.T.O)

With View: (ViewBag etc)

ASP.NET MVC - ViewBag transfers data (which is not included in model) from controller to the view.

Example: (Server code)

```
public ActionResult Index()
{ ViewBag.TotalStudents = StudentList.
  Count();
  return View(); }
```

Script: <label> Total Students: </label> @viewBag.TotalStudents

ViewData is a similar to ViewBag & is useful in transferring data from controller to view, it can contain key-value pairs (like Dictionary);

Server Code

```
Public ActionResult Index()
{ ---// List of Students
  ViewData["students"] = studentList;
  return View(); }
```

With view (@model/Viewmodel)

ASP.NET MVC use @model to use model object anywhere in the view.

```
@model Student <h2> Student Detail.</h2>
<ul>
    <li> studentId : @Model.StudentId</li>
    <li> Age : @Model.Age</li>
</ul>
```

⇒ Html helper class generates html elements using the model class object in razor view.

⇒ It binds the model object to html elements to display value of model properties into html and also assign the value of the html element to the model properties in a webform.

Ans. to the q.no-1(d)

(5)

Name of three seperate lifetimes and their differences are :-

Transient :- Each time a transient object is requested, a new instance will be created. Lightweight & stateless services.

Scoped :- Same object will be used when requested within the same request. services are created per scope / request. In a web application, every web request creates a new seperate service scope, they are created as per web request.

~~Transient :-~~

Singleton :- The same object will be used across all requests.

(P.T.O)

The singleton pattern is a software design pattern that restricts instantiation of a class to one "single" instance. Each time an instance of MySingletonServices class is asked for, it will return the same instance every time, for the lifetime of the application.

Examples:

Transient :-

```
Services.AddTransient<ITranService, Services>();
```

Singleton :-

```
Services.AddSingleton<ISingletonService, Services>();
```

Scoped :-

```
Services.AddScoped<IScopedService, Services>();
```

Ans to the q. no-2

Creating a class at Model folder

Research.cs

```
namespace Researchprog.Model
{ Public class Research
  { public int ResearchID{get;set;}
    public string ResearchTitle{ ; }
public string PrincipleInvestigator { ; }
    public string Fund { ; }
```

ResearchController.cs

```
Public IActionResult ResearchDetails()
{ var rs =new list<Models.Research>
{new Models.Research() {ResearchID=1,
  ResearchTitle ="Reduce water pollution"
Principle Investigation = "Dr. Khan";
Fund = "200,000 taka"
```

```
new Model.Research()<ResearchID=2,
ResearchTitle = "Reduce blood "

- - - - -
- - - -

// List of objects "

};
return view(rs); }
```

ResearchDetails.chtml

```
@{ViewData["Title"] = "Research info page"
{
@model IEvmerable<researchpage.Model.
    Research>
<html>
  <head> Research info </head>
<body> <br> </br>
<table> @foreach(var rsc in Model)
{ <tr> <td> ResearchID </td>
    <td@ rsc ResearchID </td> </tr>
```


 </br>

<table>
@foreach (var rsc in Model)
{ <tr> <td> Research ID </td>
<td> @rsc Research ID </td> </tr>

// script for third object

// script for fourth objects

 </br>
<table>
<body>
</html>

then run the program.

Ans to the q. no-4

(10)

Code First Approach:

Step 1:

We have to install 3 packages in VS IED

a) Microsoft.EntityFrameworkCore [V 5.0.10]

b) Microsoft.EntityFrameworkCore.SqlServer [V 5.0.10]

(c) Microsoft.EntityFramework.Cor.Tools [V 5.0.10]

Step-2

We now need to add context class, ResearchContext.cs in the Model Folder, that class will be child of DBContext.

ResearchContext.cs:

```
Using Microsoft.EntityFrameworkCore
namespace Resarch.Models
{ public class ResearchContext: DbContext
  { public ResearchContext(DbContextOptions<
    ResearchContext> option): base(options)
```

{ }

public DbSet<Research> Research{get;set;}

{ }

Step 3

Now adding ① JSON file for DB connectivity (file appsettings.json)

"ConnectionString":{

Researchconnection"="server=(local)\\

MSSQLLocalDB;database=ResearchDB;

Trusted_Connection=true;}}

Step 4

Adding entry at Startup.cs file

Public void ConfigureServices(IServiceCollectionservices)

{services.AddContentPool<Models.Research
Context>(options=>options.UseSqlServer(
Configuration.GetConnectionString("Research
Connection")));
servercies.AddControllerwithViews();
}

Step 5

Now from package manager Console

PM> Add-Migration Initialization

PM> Update-database

Ans. to the q. no- 1 (a)

Using Microsoft.AspNetCore.HTTP);

Adding code in startup.cs file

```
public void configure(IApplicationBuilder
  app, IWebHostEnvironment env)
{ app.Map("/m1, app map => {
  app.Map.Run (async Context => {
  await Context.Response.WriteAsync
  ("welcome MCS541 ASP.NET"); });
app.Run(async (Context) =>
  {await.Context.Responses.WriteApps
    (Welcome to Sub"); });
app.Map ("/M2", app => {
app.Map Run(async Context =>
```

```
{ await context.Response.WriteAsync
    ("it is from services "); }); 
app.Run (async(context)=>
    { await Context.Response.write
        Async ("1st is from service1");
    });
```