- **Explain Areas in MVC?**

  Areas in MVC are used to divide a large application into smaller functional groupings. Each area can have its own controllers, views, and models, making it easier to manage large projects by organizing related functionalities together.

- **Explain Render Section in MVC?**

  RenderSection is used in Razor views to define a section that can be overridden in child views. It allows you to create layout templates with placeholders for content that can be filled in by specific views.

  ### Example : C#

  ```
  @RenderSection("scripts", required: false)

  In the child view:

  @section scripts
          {
            <script src="path/to/script.js"></script>
          }
  ```

- **Which assembly is the MVC framework is defined?**

  The MVC framework is defined in the System.Web.Mvc assembly.

- **Apply validations using data annotation attributes.**

  Data annotations provide a way to apply validation rules to model properties.

  ### Example:

  ```
  public class Student
      {
  ```

```
    [Required]
    public string FirstName { get; set; }

    [Required]
    public string LastName { get; set; }

    [EmailAddress]
    public string Email { get; set; }

    [Phone]
    public string Mobile { get; set; }

    [Required]
    public string Gender { get; set; }

    [Required]
    public string City { get; set; }

    public string Address { get; set; }

    [Required]
    public string Course { get; set; }

    [Range(0, double.MaxValue)]
    public decimal Fees { get; set; }
}
```

- **What is difference between JavaScript and jQuery?**

  JavaScript is a programming language used to create dynamic and interactive web content.

  jQuery is a library built on JavaScript that simplifies DOM manipulation, event handling, animations, and AJAX interactions.

- **What is LINQ?**

  LINQ (Language Integrated Query) is a query syntax in C# that allows querying of data from various data sources (collections, databases, XML, etc.) in a consistent manner.

- **What is difference between LINQ and Entity Framework.?**

  LINQ is a query language that can be used to query collections in memory (like lists, arrays, etc.).

  Entity Framework (EF) is an ORM (Object Relational Mapper) that allows you to work with a database using .NET objects, and it supports LINQ to Entities for querying the database.

- **Create One Registration Page for Student Registration Model Class and When Student Click on Submit Button, Details of student must be display on same page. Fields : First Name, Last Name, Email id, Mobile, Gender, City, Address, Course (Dropdown list), Fees, Tax (Label with Text 12.50%), Total Fees.**

  ## Model: C#

```
public class Student
{
   [Required]
   public string FirstName { get; set; }

   [Required]
   public string LastName { get; set; }

   [EmailAddress]
   public string Email { get; set; }

   [Phone]
   public string Mobile { get; set; }
```

```
    [Required]
    public string Gender { get; set; }

    [Required]
    public string City { get; set; }

    public string Address { get; set; }

    [Required]
    public string Course { get; set; }

    [Range(0, double.MaxValue)]
    public decimal Fees { get; set; }

    public decimal Tax => 0.125m;

    public decimal TotalFees => Fees + (Fees * Tax);
}
```

Controller: C#

```
public ActionResult Register()
{
    return View();
}

[HttpPost]
public ActionResult Register(Student student)
{
    if (ModelState.IsValid)
```

```
    {
        ViewBag.Message = "Student registered successfully!";
        return View(student);
    }
    return View(student);
}
```

View: Html

```
@model StudentRegistration.Models.Student
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    <div>
        @Html.LabelFor(m => m.FirstName)
        @Html.TextBoxFor(m => m.FirstName)
        @Html.ValidationMessageFor(m => m.FirstName)
    </div>
    <!-- Add other fields similarly -->
    <div>
        @Html.Label("Tax: 12.50%")
    </div>
    <div>
        @Html.LabelFor(m => m.TotalFees)
        @Html.TextBoxFor(m => m.TotalFees, new { @readonly = "readonly"
})
    </div>
    <button type="submit">Submit</button>
}
```

- **Create View Model for CRUD Operation in Country, State and City Table with validation using EF.**

Model: C#

```
public class Country
{
   public int CountryId { get; set; }
   [Required]
   public string Name { get; set; }
}

public class State
{
   public int StateId { get; set; }
   [Required]
   public string Name { get; set; }
   public int CountryId { get; set; }
}

public class City
{
   public int CityId { get; set; }
   [Required]
   public string Name { get; set; }
   public int StateId { get; set; }
}
```

View Model: C#

```
public class LocationViewModel
{
   public int CountryId { get; set; }
   public string CountryName { get; set; }

   public int StateId { get; set; }
   public string StateName { get; set; }

   public int CityId { get; set; }
   public string CityName { get; set; }
}
```

- **What is Data Annotation Validator Attributes in MVC?**

  Data annotation attributes are used to enforce validation rules on model properties. Examples include [Required], [StringLength], [Range], [EmailAddress], etc.

- **What are AJAX Helpers in MVC?**

  AJAX helpers in MVC are used to make asynchronous requests to the server from the client. Examples include Ajax.BeginForm, Ajax.ActionLink, and AjaxOptions.

- **Write a program to perform join on two lists.**

  C#

```
var list1 = new List<int> { 1, 2, 3 };
var list2 = new List<int> { 3, 4, 5 };
```

```
var joinedList = from l1 in list1
                 join l2 in list2 on l1 equals l2
                 select new { l1, l2 };

foreach (var item in joinedList)
{
   Console.WriteLine($"{item.l1} - {item.l2}");
}
```

- **Perform CRUD operation using LINQ**

Read: C#

```
var students = db.Students.ToList();
```

Create: C#

```
db.Students.Add(new Student { FirstName =    "John", LastName =
"Doe" });
db.SaveChanges();
```

Update: C#

```
var student = db.Students.FirstOrDefault(s => s.StudentId == 1);
if (student != null)
{
   student.FirstName = "UpdatedName";
   db.SaveChanges();
}
```

Delete: C#

```
var student = db.Students.FirstOrDefault(s => s.StudentId == 1);
if (student != null)
{
   db.Students.Remove(student);
   db.SaveChanges();
}
```

- **What are required attributes for Ajax call?**

  url: The URL to which the request is sent.

  type: The type of request (GET, POST, etc.).

  data: The data to be sent to the server.

  success: A callback function to be executed if the request succeeds.

- **Create insert/update/delete/view on same view using AJAX**

  Html

```
@model StudentRegistration.Models.Student
@using (Html.BeginForm("Register", "Student", FormMethod.Post, new { id
= "studentForm" }))
{
   @Html.AntiForgeryToken()
   @Html.TextBoxFor(m => m.FirstName)
   <button type="button" onclick="submitForm()">Submit</button>
}

<div id="result"></div>

@section scripts {
   <script>
      function submitForm() {
         $.ajax({
            url: '@Url.Action("Register")',
            type: 'POST',
            data: $('#studentForm').serialize(),
            success: function (response) {
               $('#result').html(response);
            }
         });
      }
   </script>
}
```

- **What is JsonResultType in MVC?**

  JsonResult is a type of ActionResult in MVC that returns JSON-formatted data. It is typically used for AJAX requests.

  C#

```
public JsonResult GetStudent(int id)
{
   var student = db.Students.Find(id);
   return Json(student, JsonRequestBehavior.AllowGet);
}
```

- **What is the meaning of Unobtrusive JavaScript?**

  Unobtrusive JavaScript separates JavaScript code from HTML markup. It uses data-* attributes in HTML elements to store event handlers and other data, allowing for cleaner and more maintainable HTML.

- **What are the sub types of ActionResult?**

  ViewResult

  PartialViewResult

  RedirectToRouteResult

  RedirectResult

  JsonResult

  FileResult

  ContentResult

  EmptyResult

- **Get Data using FormCollection and store data into session and show on another view.**

Controller: C#

```
[HttpPost]
public ActionResult Submit(FormCollection form)
{
   var firstName = form["FirstName"];
   var lastName = form["LastName"];

   Session["FirstName"] = firstName;
   Session["LastName"] = lastName;

   return RedirectToAction("Display");
}

public ActionResult Display()
{
   ViewBag.FirstName = Session["FirstName"];
   ViewBag.LastName = Session["LastName"];
   return View();
}
```

View: Html

```
<div>
  First Name: @ViewBag.FirstName
</div>
<div>
   Last Name: @ViewBag.LastName
</div>
```

- **Bind Following Controls using Model Class Dynamically DropDownList, ListBox, CheckBoxList**

Model: C#

```
public class Student
{
   public int StudentId { get; set; }
   public string Name { get; set; }
}
```

Controller: C#

```
public ActionResult Register()
{
   var students = new List<Student>
   {
      new Student { StudentId = 1, Name = "John" },
      new Student { StudentId = 2, Name = "Jane" }
   };

   ViewBag.Students = new SelectList(students, "StudentId", "Name");
   return View();
}
```

View: Html

```
@Html.DropDownList("StudentId", (SelectList)ViewBag.Students)
@Html.ListBox("StudentId", (SelectList)ViewBag.Students)
@Html.CheckBoxList("StudentId", (SelectList)ViewBag.Students)
```

- **Different way to return view in MVC?**

```
return View();

return View("ViewName");

return View(model);

return PartialView();
```

```
return PartialView("ViewName");

return RedirectToAction("ActionName");
```

- **Create Multiple Image Upload Example, Image Display after Upload. (Use Database table for save data like image name, size, type, URL)**

Model: C#

```
public class Image
{
   public int ImageId { get; set; }
   public string ImageName { get; set; }
   public string ImageUrl { get; set; }
   public long ImageSize { get; set; }
   public string ImageType { get; set; }
}
```

Controller: C#

```
public ActionResult Upload()
{
   return View();
}

[HttpPost]
public ActionResult Upload(IEnumerable<HttpPostedFileBase> files)
{
   foreach (var file in files)
   {
      if (file != null && file.ContentLength > 0)
      {
         var image = new Image
         {
            ImageName = Path.GetFileName(file.FileName),
            ImageType = file.ContentType,
            ImageSize = file.ContentLength,
            ImageUrl = Path.Combine(Server.MapPath("~/Images"),
Path.GetFileName(file.FileName))
         };

         file.SaveAs(image.ImageUrl);
```

```
            db.Images.Add(image);
            db.SaveChanges();
        }
    }

    return RedirectToAction("Index");
}

public ActionResult Index()
{
    var images = db.Images.ToList();
    return View(images);
}
```

View: Html

```
@using (Html.BeginForm("Upload", "Home", FormMethod.Post, new {
enctype = "multipart/form-data" }))
{
    <input type="file" name="files" multiple />
    <button type="submit">Upload</button>
}

@foreach (var image in Model)
{
    <div>
        <img src="@Url.Content("~/Images/" + image.ImageName)"
alt="@image.ImageName" />
        <p>@image.ImageName (@image.ImageSize bytes)</p>
    </div>
}
```