MVC

ASP.NET:-

DEFAULT.ASPX DEFAULT.ASPX.CS

Tightly coupled

ASP.NET:- Loosely coupled.

Model:- Model helps to create **Classes**(C#). And it is used to right **Business Logic** in it.

View:- It creates file of ".CSHTML".

Controller:- It is a Request Handler.

Day18

MVC:- It is an Architecture. MVC follows three Tier Architecture….

1. 2-Tier Architecture.
2. 3-Tier Architecture.
3. N-Tier Architecture.

These 3 Tiers are Project Handler.

2-Tier Architecture:-

When you run a direct database on a webpage, it is called 2-Tier Architecture.

3-Tier Architecture:-

It's a predefined Architecture.

N-Tier Architecture:-

When more than 3 layers are added in project is known as N-Tier Architecture.

HTTPS:- HTTPS stands for Hyper Text Transfer Protocol Secure. It is a combination of the Hypertext Transfer Protocol (HTTP) with the Secure Socket Layer (SSL). It allows you to communicate securely with the web server.

Create MVC Project:-

- Open Visual Studio.

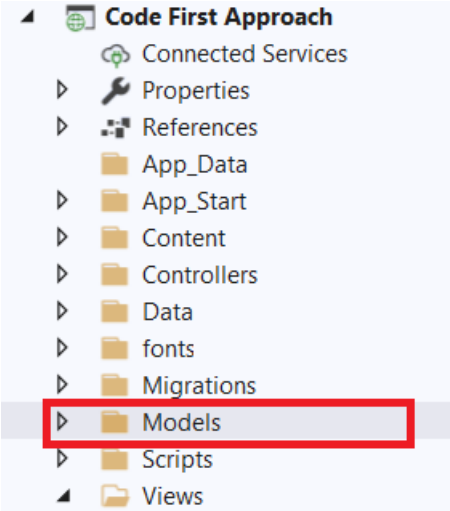
- Create New Project.
- Select ASP.NET web application.
- Set project name “with MVC”.
- Select location to save.
- Select MVC template.
- Now click on create button.
- Go to Solution Explorer.
    - **References** includes all Library.
    - **App_Data** is not for used.
    - **App_Start** includes configuration files.
        - BundelConfig.CS:- To create Bundles.
        - FilterConfig.CS:- These filters are applied to all actions and controllers.
        - RouteConfig.CS:- It is used to set routing for **the application**.

    - **Content** folder includes Designing files.
    - **Controllers** control project handling. It is Class.
    - **Fonts**
    - **Models**
    - **Scripts** include all JS files.
    - **Views** have GUI.
        - **Shared** This folder shows those things which are there in many places.
            - **_Layout.cshtml** is a Master File.
    - **favicon.ico** used for changing the Icon.

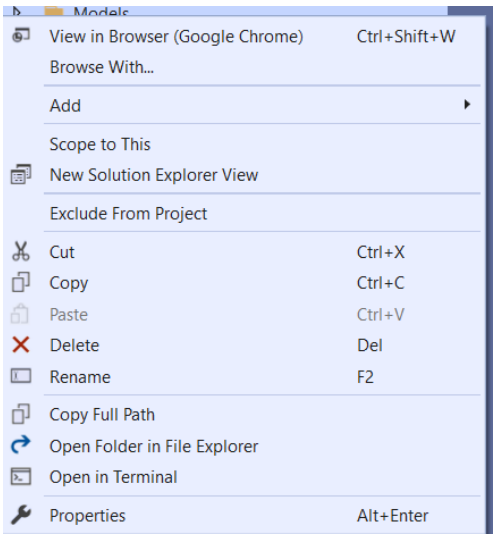
- **Global.asax** This file is only one in one project. It is used for manage **Application Level** & **Session Level Events**.
- **packages.config** used in some project types to maintain the list of packages referenced by the project.

How to Create <u>Model</u>?

- Right Click on Models Folder.



- Choose Add.



- Select Code then Class.

- Set a name “Employee”.
- Click on Add Button.

```
namespace WebApplication4_with_MVC.Models
{
    public class Employee
    {   prop
        public int Id { get; set; }
        public string Name { get; set; }
        public int Salary { get; set; }
        public string Address { get; set; }
    }
}
```

Enter Tab Button 2 Times.

Now go to <u>Controller</u>?

- Right click on Controller.
- Click on Add then **Controller**.
- Select controller then **MVC 5 Controller**.
- Click on **Add** button.
- Set Controller Name “EmpController”.

```
namespace WebApplication4_with_MVC.Controllers
{
    public class EmployeeController : Controller
    {
        // GET: Employee
        public ActionResult Index()
        {
            var EmpInDb = _context.employees.ToList();
            return View(EmpInDb);
        }

    }
}
```

Now Create Model View Controller

- Create new Project.
- Select **ASP.NET(Web Application)** Templet.
- Set name as "with MVC".
- Select **MVC** templet.
- Then click on **Create** button.
- Now go to **Solution Explorer**.
- Right click on **Models** folder.
- Select **Add** option then **New Item**.
- Choose 'Code'.
- Set Name as "Employee".

```
namespace WebApplication4_with_MVC.Models
{
    public class Employee
    {   prop
        public int Empno { get; set; }
        public string Name { get; set; }
        public string Address { get; set; }
        public int Salary { get; set; }
    }
}
```

Enter Tab Button 2 Times.

Controller:-

- Right click on **Controller** folder.
- Choose **Add**.
- Then click on **Controller** option.

- Select **MVC 5 Controller-Empty**.
- Then click on **Add** button.
- Set name as "**Emp**" with **Controller** Keyword.

```
namespace WebApplication4_with_MVC.Controllers
{
    public class EmpController : Controller
    {
        // GET: Emp
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

All the action results are there with all the views.

Now Create View:-

- Right click on View.
- Select Add View.
- Choose "MVC5 View" template.
- Press Add Button.
- Set name as "Index".
- Click on Add Button.

Result:-

Inside the view a file will be created with the name of the 'Index.cshtml' in the emp folder.

- Now go to Model folder.
- Then Employee Class.

```
namespace WebApplication_with_MVC.Controllers
{
    public class EmpController : Controller
    {
        // GET: Emp
        public ActionResult Index()
        {
            Employee employee = new Employee()
            {
                Empno = 101,
                Name = "Amit",
                Address = "Chandigarh",
                Salary = 90000
            };
```

Or

```
Var employee= new Employee()
{s
     Empno = 101,
     Name = "Amit",
     Address = "Chandigarh",
     Salary = 90000

};
```

Or

```
Employee employee = new Employee();
employee.Empno = 101;
employee.Name = "Amit";
employee.Address = "Chandigarh";
employee.Salary = 90000;
```

Or

```
Employee employee = new Employee{ Empno = 101, Name = "Amit",
Address = "Chandigarh", Salary = 90000};
```

```
            return View(employee);
        }
    }
}
```

- Now coding for View.

```
@model WebApplication_with_MVC.Models.Employee
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
<div>
    <b>Empno.</b>@Model.Empno<br />
    <b>Name</b>@Model.Name<br />
    <b>Address</b>@Model.Address<br />
    <b>Salary</b>@Model.Salary<br />
</div>
```

- Now do Coding to show multiple Values in Web.

```
namespace WebApplication_with_MVC.Controllers
{
  public class EmpController : Controller
    {
      // GET: Emp
      public ActionResult Index()
       {
         var employee = new List<Employee>
          {
            new Employee{Empno=101, Name = "Amit", Address
= "Chd", Salary = 90000 },
```

```
            new Employee{Empno=102, Name= "Sarita", Address
= "Chd", Salary = 80000 },
            new Employee{Empno=103, Name = "Rajan", Address
= "Himachal", Salary = 54154 }
          };
            return View(employee);
        }
    }
}
```

- Go to Index View.

```
@model IEnumerable<WebApplication_with_MVC.Models.Employee>
@{
    ViewBag.Title = "Index";
}

<h2>Employee Detail</h2>
@foreach (var item in Model)
{
    <div>
        <b>Empno.</b>@item.Empno<br />
        <b>Name</b>@item.Name<br />
        <b>Address</b>@item.Address<br />
        <b>Salary</b>@item.Salary<br />
    </div>
}
```

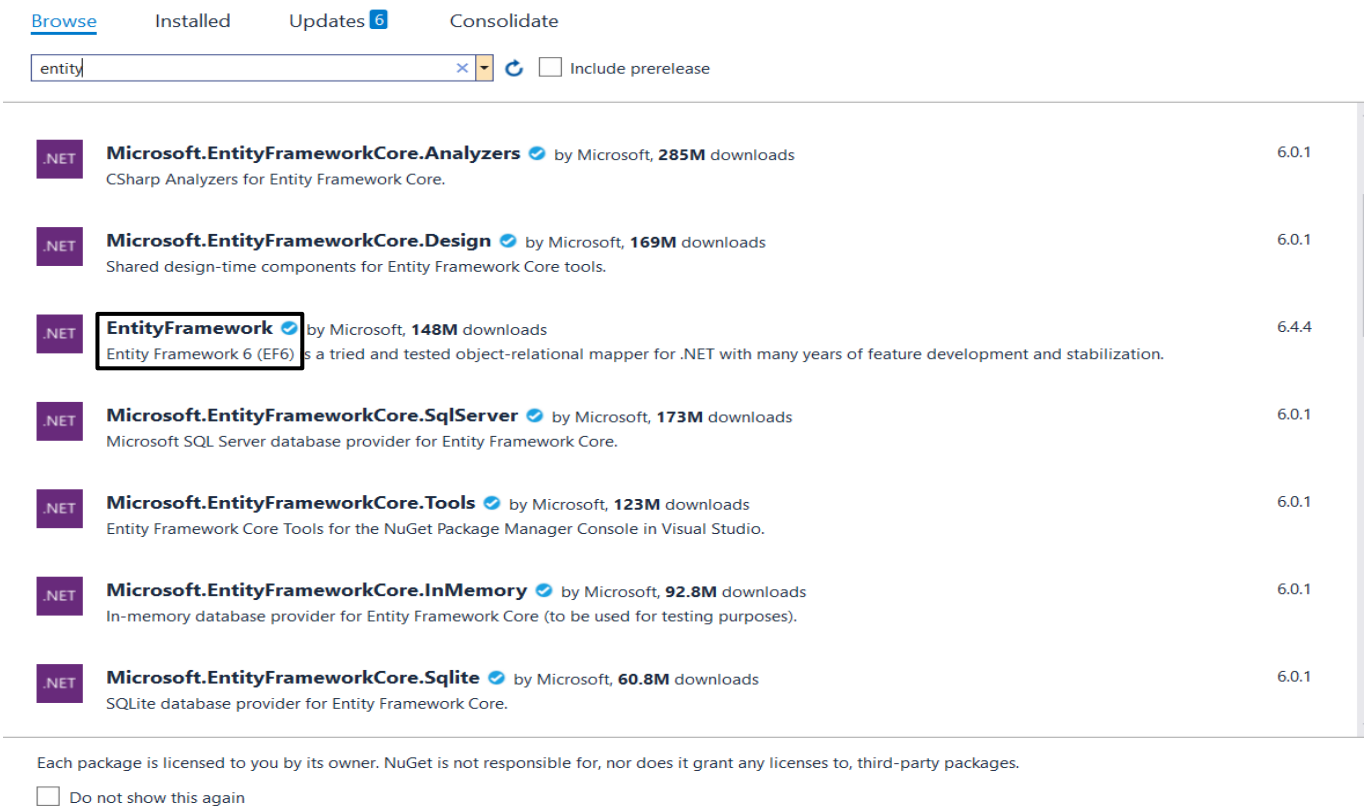
Now run it.

Day21

Entity FrameWork with Code First Approach

How to Create DataBase with Code First Approach:-

- Download **Entity Framework**
  - ✓Go to **Tools** Bar.
  - ✓Select **NuGet Package Manager**.
  - ✓Then click on **Manage NuGet Packages For Solution….**
  - ✓A window will be appear.
  - ✓Search Entity Framework.



✓And install it in your project.

- Go to Model folder.
- Right click on it then Add New Class(Visual C#).
- Set name as “Student”.
- Click on Add Button.

```
namespace WebApplication_EntityFrameWork.Models
{
    public class Student
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Email { get; set; }
        public int Age { get; set; }
    }
}
```

- Open web.config File.

Note:- We can create connection where we want after “**</appsettings>**” Tag.

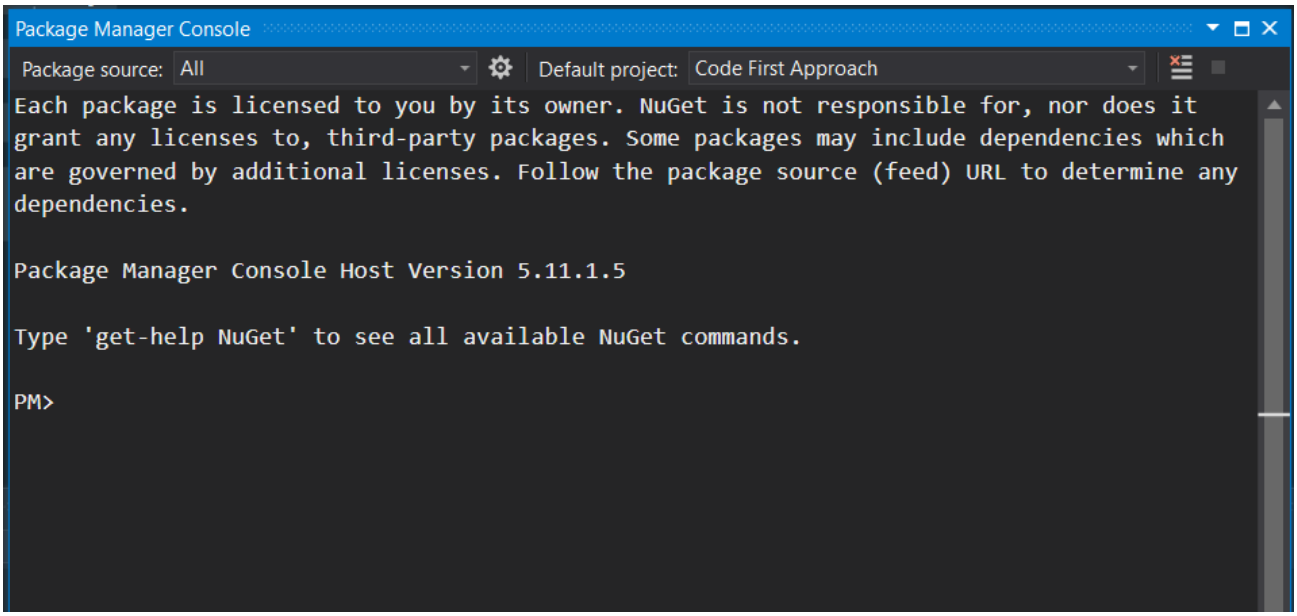
```
<connectionStrings>
<add name="constr" connectionString ="server=Jarvis
\Sqlexpress; database=dbstudent11; integrated security
=true" providerName= "System.Data.SqlClient" />
</connectionStrings>
```

- Right click on Project from **Solution Explorer**.
- Choose Add option.
- Create folder as “Data”.
- Right click on **Data** folder.
- Choose **Add** then **Class** option.
- Set name “**ApplicationDbContext**”.
- Click on **Add** Button.

```
namespace WebApplication_EntityFrameWork.Data
{
    public class ApplicationDbContext:DbContext
    {
        public ApplicationDbContext():base("constr")
        {

        }
        public DbSet<Student> students { get; set; }
    }
}
```

- Now go to **Tools** Tab.
- Click on **NuGet Package Manager**.
- Then **Package Manager Console**.



- Now right commands in its window.

  ✓ enable-migrations(this is for one time in one project).

  ✓ add-migration initLoad(migration name is depend on your choice).

- ✓ update-database (whenever you add-migration don't forget to update database).

- Now go to **Server Explorer**.
- Right click on Data Connection.
- Go to Table folder.
- You'll see there is a Table has Created named with '**Student**'.

Create Another Table In Database named with Subjects

- Right click on **Model** Folder.
- Choose **Add** then **Class**.
- Set name as "**Subject**".

```
namespace WebApplication_EntityFrameWork.Models
{
    public class Subject
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
```

- Then

```
public DbSet<Subject> subjects { get; set; }
```

- Then
- Add migrations in Package Manager Console window from NuGet Package Manager in Tools Tab.
    - add-migration AddSubjectTable.
    - update-database.

If you want to Add Column in Subject Table

- Go to Model.
- Choose Subject File(Class).

```
public class Subject
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public bool Status { get; set; }
    }
```

- add-migration AddStatusColumnToSubjectTable
- update-database.

Day 22

How to Create Controller?

- Right click on **Controller**.
- Select **Add** option then **Controller**.
- Select **MVC5** template then click on **Add** button.
- Set controller name as "**StudentController**".

```
namespace WebApplication_EntityFrameWork.Controllers
{
  public class StudentController : Controller
   {
     private readonly ApplicationDbContext _context;
      public StudentController()
       {
         _context = new ApplicationDbContext();
       }
       protected override void Dispose(bool disposing)
       {
        _context.Dispose();
       }
      // GET: Student1
       public ActionResult Index()
       {
        var studentindb = _context.students. ToList();
        return View(studentindb);
       }
   }
}
```

- Now **Add a View**.
- Choose **Add** button.
- Set name as "Index".

- Set templet as “List”.
- Model Class-WebApplication_EntityFrameWork .Models .
- DataContext Class- ApplicationDbContext.
- Now click Add Button.

Now run the file.

```
@model IEnumerable<Code_First_Approach.Models.NewStudent>

<p>
    @Html.ActionLink("Create New", "Create")
</p>
@if (!Model.Any())
{
    <h2>No Data Found</h2>
}
else
{
    <table class="table">
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.subject.Name)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Name)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Email)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Age)
            </th>
            <th>Actions</th>
        </tr>
<tbody>
 @foreach (var item in Model)
  {
    Using (Html.BeginForm(“Delete_Rec”,“Student”, new{id=item.Id}))
      {
       <tr>
          <td>
              @Html.DisplayFor(modelItem => item.subject.Name)
```

```
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Name)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Email)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Age)
            </td>
            <td>
             @Html.ActionLink("Edit", "Edit", new { id = item.Id }) |
        @Html.ActionLink("Details", "Details", new { id = item.Id }) |
        <input type="submit" class="btn-link" value="Delete"
onclick="return confirm('want to delete this file')" />
            </td>
           </tr>
        }
          }
        </tbody>
      </table>
}
```

Now Run It.

Now do coding for another link **Create**

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

- Right click on action and **Add a View**.
- Select **MVC5** templet.
- Set name as "**Create**".
- Templet – Create.
- Model Class – Same as Index.
- Data Context class - Same as Index.
- Click o **Add** Button.

- Now run program.

Now go to controller:-

```
[HttpPost]
        public ActionResult Create(Student student)
        {
            if (student == null)
                return HttpNotFound();
            _context.students.Add(student);
            _context.SaveChanges();
            return RedirectToAction("Index");
        }
```

Now run Program……

Day23

To add link in Navigation Bar:-

1. Go to **Solution Explorer**.
2. Then **Views** Folder.
3. Then **Shared** folder.
4. Then choose "**_Layout.cshtml**" file and open it.

```
<ul class="nav navbar-nav">
    <li>@Html.ActionLink("Home", "Index", "Home")</li>
    <li>@Html.ActionLink("About", "About", "Home")</li>
    <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
    <li>@Html.ActionLink("Student List","Index","Student1")</li>
</ul>
```

Coding for Edit Button:-

1. Go to **Student** Controller.
2. Create an Action.
3. Set name as "**Edit**".

```
public ActionResult Edit(int id)
        {
            if (id == 0)
                return HttpNotFound();
            var studentInDb = _context.students.Find(id);
            if (studentInDb == null)
                return HttpNotFound();
            return View(studentInDb);
        }
```

Or

```
public ActionResult Edit(int id)
 {
  if (id == 0)
  return HttpNotFound();
  var studentInDb =_context.students.FirstOrDefault(s=>s.id==Id);
  if (studentInDb == null)
  return HttpNotFound();
  return View(studentInDb);
 }
```

Or

```
public ActionResult Edit(int id)
 {
  if (id == 0)
  return HttpNotFound();
  var studentInDb =_context.students.where(s=>s.id==Id).
FisrtOrDefault();
  if (studentInDb == null)
  return HttpNotFound();
  return View(studentInDb);
 }
```

- Right click on action and **Add a View**.
- Select **MVC5** templet.
- Set name as "**Edit**".
- Templet – Edit.
- Model Class – Same as Index.
- Data Context class - Same as Index.
- Click o **Add** Button.
- Now run program.

<u>Now run Program.</u>

Coding for Post Update:-

```
[HttpPost]
        public ActionResult Edit(Student student)
        {
            var studentFromDb = _context.students.Find(student.Id);
            if (studentFromDb == null)
                return HttpNotFound();
            studentFromDb.Name = student.Name;
            studentFromDb.Age = student.Age;
            studentFromDb.Email = student.Email;
            studentFromDb.SubjectId = student.SubjectId;
            _context.SaveChanges();
            return RedirectToAction(nameof(Index));
        }
```

Do coding for Details:-

```
public ActionResult Details(int id)
        {
            if (id == 0)
                return HttpNotFound();
            var studentFromDb = _context.students.Find(id);
            if (studentFromDb == null)
                return HttpNotFound();
            return View(studentFromDb);
        }
```

- Right click on action and **Add a View**.
- Select **MVC5** templet.
- Set name as "**Details**".
- Templet – Details.
- Model Class – Same as Index.
- Data Context class - Same as Index.

- Click o **Add** Button.
- Now run program.

Coding for Delete:-

```
public ActionResult Delete(int id)
        {
            var studentdel = _context.students.Find(id);
            if(studentdel==null)
                return HttpNotFound();
            return View(studentdel);
        }


[HttpPost]
        public ActionResult Delete_Rec(int id)
        {
            var studentFromDb = _context.students.Find(id);
            if (studentFromDb == null)
                return HttpNotFound();
            _context.students.Remove(studentFromDb);
            _context.SaveChanges();
            return RedirectToAction(nameof(Index));
        }
```

- Right click on action and **Add a View**.
- Select **MVC5** templet.
- Set name as "**Delete**".
- Templet – Delete.
- Model Class – Same as Index.
- Data Context class - Same as Index.
- Click o **Add** Button.
- Now run program.

Routing

Routing:- It is a URL.

Routing has two types:-

i) Conventional Routing
ii) Attribute Base Routing

It is Old Type.

It is New Type.

- Create new Project.
- ASP.NET Application (Templet).
- Set name as "**_Routing**".
- MVC(Template).
- Then create.
- Right click on Controller.
- Add new controller **MVC-5**.
- Set name as "TestController".
- Create View.

Go to RouteConfig file

Create New Project:-

- Set name as "_Code_First_Validation".
- Click on **Add** button.
- Add **.Net FrameWork**.
- Go to Google then Search "https://bootswatch.com/".
  a. Select **Version3**.
  b. Choose any theme.
  c. Now selected theme opens a Window.
  d. Click on "**bootstrap.css**" option.
  e. Save this file.
  f. Copy this file.
- Now open **Visual Studio**.
- Go to **Content** folder in **Solution Explorer**.
- Then paste the file that copied.
- Now copy its file name.
- Go to "**App_Start**" folder.
- Choose "**BundelConfig.CS**" file and open it.

```
bundles.Add (new
StyleBundle("~/Content/css").Include("~/Content/bootstrap_Readable.
css","~/Content/site.css"));
```

- Install **Entity FrameWork**.
- Create a **Connection String** in **Web.config** File after

  `</appSettings>`.

```
<connectionStrings>
    <add name="consta" connectionString="server=Jarvis\sqlexpress;
database=dbregister_cntstatcty0; integrated security=true"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

- Now create a model.
- Set name as "**Student**".

```
public class Student
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int Age { get; set; }
        public string Email { get; set; }
    }
```

- Now create a folder & set a name "**DataContext**".
- Create a class in "**DataContext**" folder.
- Set name as "**ApplicationDbContext**".

```
public class ApplicationDbContext:DbContext
    {
        public ApplicationDbContext():base("consta")
        {

        }
        public DbSet<Student> students { get; set; }
    }
```

Add NameSpace

- Go to Tools Tab.
- Click on **NuGet Package Manager**.
- Then click on **Package Manager Console**.
- enable-migrations.
- add-migration initload.
- update-database.

Go to Controller:-

- Create a controller.
- Set name as “StudentController”.

```
public class StudentController : Controller
    {
      private readonly AppDbContext _hii;
      public StudentController()
      {
          _hii = new AppDbContext();
      }
      protected override void Dispose(bool disposing)
      {
          _hii.Dispose();
      }
      // GET: Student
      public ActionResult Index()
      {
          var v1 = _hii.students.ToList();
          return View(v1);
      }
    }
```

- Create a view.
- Set name as “Index”.
- Now go to **Solution Explorer** then **Views** folder and then **Shared** folder and in last “**_Layout.cshtml**” file.

```
<ul class="nav navbar-nav">
   <li>@Html.ActionLink("Home", "Index", "Home")</li>
   <li>@Html.ActionLink("About", "About", "Home")</li>
   <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
   <li>@Html.ActionLink("Student Rec.","Index","Student")</li>
</ul>
```

- Go to **Index.cshtml** file.

```
@model IEnumerable<Full_practise.Models.Student>
@{
    ViewBag.Title = "Index";
}

<h2>Student List</h2>
@if (!Model.Any())
{
    <h1 class="text-center text-capitalize text-success"
style="font-family:'Times New Roman', Times, serif">No Data
Found</h1>
}
else
{
<table class="table table-bordered table-responsive table-hover"
style="border:dotted">
    <thead style="backdrop-filter:blur(50px)")">
        <tr>
            <th>
                Name
            </th>
            <th>
                Age
            </th>
            <th>
                Email
            </th>
            <th>
                Actions
            </th>
        </tr>
    </thead>
    @foreach (var item in Model)
    {
        using (Html.BeginForm("Delete", "Student", new { id =
item.Id }))
        {
            <tbody style="background-color:gold">
                <tr>
                    <td>
                        @item.Name
                    </td>
                    <td>
```

```
                        @item.Age
                    </td>
                    <td>
                        @item.Email
                    </td>
                    <td>
                        @Html.ActionLink("Edit", "Edit", new { id =
item.Id })|
                        @Html.ActionLink("Detail", "Detail", new {
id = item.Id })|
                        <input value="Delete" type="submit"
onclick="return confirm('Want to delete this data???')" />
                    </td>
                </tr>
            </tbody>
        }
    }
</table>
}
@Html.ActionLink("Create New", "Create", "Student", new
{@class="btn btn-warning"})
```

- Go to **Controller**.

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

- Now **Add a View**.
- Set name as “Create”.

```
@model Full_practise.Models.Student
@{
    ViewBag.Title = "Create";
}

<h2>Enter Student Record</h2>
@using (Html.BeginForm())
{
  @Html.ValidationSummary(false, "", new {@class="alert-danger"})
  <div class="badge">
     @Html.LabelFor(s => s.Name)
```

```
      @Html.TextBoxFor(s => s.Name, new { @class = "form-control" })
   </div>
   <div class="badge">
      @Html.LabelFor(s => s.Age)
      @Html.TextBoxFor(s => s.Age,new { @class = "form-control" })
   </div>
   <div class="badge">
      @Html.LabelFor(s => s.Email)
      @Html.TextBoxFor(s => s.Email,new { @class = "form-control" })
   </div>
   <div>
      <input type="submit" value="Save" class="btn btn-block btn-
success" />
      @Html.ActionLink("Back to List","Index")
   </div>
}
```

- Go back to **Controller**.

```
[HttpPost]
   public ActionResult Create(Student student)
   {
       if (student == null)
           return HttpNotFound();
       if (!ModelState.IsValid)
           return View();
var duplicate = _hii.students.FirstOrDefault(s => s.Email ==
student.Email);
       if(duplicate!=null)
         {
           ModelState.AddModelError("Email", "Email already Used");
            return View();
         }
         _hii.students.Add(student);
         _hii.SaveChanges();
            return RedirectToAction(nameof(Index));
   }
```

- Now Run it.

Do Coding for Edit Link:-

```
public ActionResult Edit(int id)
        {
            if (id == 0)
                return HttpNotFound();
            var v3 = _hii.students.Find(id);
            if (v3 == null)
                return HttpNotFound();
            return View(v3);
        }
```

- Now **Add a View**.
- Set name as "**Edit**".

```
@model Full_practise.Models.Student
@{
    ViewBag.Title = "Edit";
}

<h2>Edit Data</h2>
@using (Html.BeginForm())
{
  @Html.ValidationSummary(false, "", new {@class="alert-danger"})
  <div class="badge">
    @Html.LabelFor(m => m.Name)
    @Html.TextBoxFor(m => m.Name, new { @class = "form-control" })
  </div>
  <div class="badge">
    @Html.LabelFor(m => m.Age)
    @Html.TextBoxFor(m => m.Age, new { @class = "form-control" })
  </div>
  <div class="badge">
    @Html.LabelFor(m => m.Email)
    @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })
  </div>
  <div>
    <input type="submit" value="Update" class="btn btn-success" />
    @Html.ActionLink("Back to list", "Index", "Student")
  </div>
}
```

- Now go back to **Controller**.

```
[HttpPost]
        public ActionResult Edit(Student student)
        {
            if (student == null)
                return HttpNotFound();
            var v4 = _hii.students.Find(student.Id);
            if (v4 == null)
                return HttpNotFound();
            v4.Name = student.Name;
            v4.Age = student.Age;
            v4.Email = student.Email;
            _hii.SaveChanges();
            return RedirectToAction(nameof(Index));
        }
```

- Now Run it.

Do Coding for Delete Link:-

- Go to **Controller**.

```
public ActionResult Delete(int id)
        {
            if (id == 0)
                return HttpNotFound();
            var v6 = _hii.students.Find(id);
            if (v6 == null)
                return HttpNotFound();
            _hii.students.Remove(v6);
            _hii.SaveChanges();
            return RedirectToAction(nameof(Index));
        }
```

- Go to **Index File**.

```
@foreach (var item in Model)
 {
  using (Html.BeginForm("Delete", "Student", new { id = item.Id }))
      {
        <tbody style="background-color:gold">
          <tr>
            <td>@item.Name</td>
            <td>
                @item.Age
            </td>
            <td>
                @item.Email
            </td>
            <td>
        @Html.ActionLink("Edit", "Edit", new { id = item.Id })|
        @Html.ActionLink("Detail", "Detail", new { id = item.Id })|
        <input value="Delete" type="submit" onclick="return confirm
('Want to delete this data???')" />
            </td>
            </tr>
            </tbody>
        }
    }
```

For validation:-

- Go to **Details.cshtml** file.

```
@{
    ViewBag.Title = "Details";
    var email = Model.Email == null ? "No Email Found" :
Model.Email;
}
```

Now replace `<td>@Model.Email</td>` into `<td>@email</td>`.

Types of Validation:-

I) Client-Side Validation
II) Server-Side Validation
III) Custom Validation

Client-Side validation:-

Create Validation for Create.cshtml file:-

Open Create.cshtml File/View.

```
@using (Html.BeginForm())
{
 @Html.AntiForgeryToken()
 @Html.ValidationSummary(false, "", new { @class = "text-danger" })
 <div class="form-group">
     @Html.LabelFor(m => m.Name)
     @Html.TextBoxFor(m => m.Name, new { @class = "form-control" })
 </div>

 <div class="form-group">
     @Html.LabelFor(m => m.Age)
     @Html.TextBoxFor(m => m.Age, new { @class = "form-control" })
 </div>

 <div class="form-group">
    @Html.LabelFor(m => m.Email)
```

```
    @Html.TextBoxFor(m => m.Email, new { @class = "form-control" })
 </div>

 <div class="form-group">
     <input type="submit" value="Save" class="btn btn-success" />
 </div>
}
<div>
    @Html.ActionLink("Back To List", "Index")
</div>
```

Note:- ValidationSummary and ValidationMessageFor are used only to show the message of validation.

- Now open BundelConfig.cs file.

Go to "`bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include("~/Scripts/jquery.validate*"));`".

- Copy and paste this text in Create.cshtml file.

```
@section scripts
{
    @Scripts.Render("~/bundles/jqueryval")
}
```

Now run it.

- Now open Student Model file.

```
public class Student
    {
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
        [Required]
        public string Email { get; set; }
        [Required]
        public int? Age { get; set; }
    }
```

This Key is used for restricting Empty Fields.

This is used for allowing null value.

- Now add-migration.
- And update-database.

For Custom Message in Validation:-

```
public class Student
    {
        public int Id { get; set; }
        [Required(ErrorMessage = "Name Empty")]
        public string Name { get; set; }
        [Required(ErrorMessage = "Please Enter an Email")]
        [EmailAddress]
        public string Email { get; set; }
        [Required]
        [Range(18, 40, ErrorMessage = "Enter Valid Age")]
        public int Age { get; set; }
    }
```

If you want to change Headings Name without disturbing Cshtml files:-

```
public class Student
    {
        public int Id { get; set; }
        [Display(Name = "Student Name")]
        public string Name { get; set; }
        [Display(Name = "Student Email")]
        public string Email { get; set; }
    }
```

If you want to put Validation on Edit file:-

Repeat in this file as you did in the "**Create.cshtml**" file.

How to disable JavaScript in browser:-

- Go to **Browser**.
- Right click on Screen.
- Choose **inspect** option.
- Go to **settings** icon.
- Look for **Disable JavaScript** option and click on it.

Server-Side Validation:-

- Go to **Student Controller**.

```
[HttpPost]
        public ActionResult Create(Student student)
        {
            if (student == null)
                return HttpNotFound();
            //Validation Server Side
            if (!ModelState.IsValid)
                return View();
            _context.students.Add(student);
            _context.SaveChanges();
            return RedirectToAction("Index");
        }
```

For Edit

```
[HttpPost]
        public ActionResult Edit(Student student)
        {
            if (student == null)
                return HttpNotFound();
            var studentFromDb = _context.students.Find(student.Id);
            if (studentFromDb == null)
                return HttpNotFound();
            if (!ModelState.IsValid)
                return View(studentFromDb);
            studentFromDb.Name = student.Name;
            studentFromDb.Age = student.Age;
            studentFromDb.Email = student.Email;
            studentFromDb.SubjectId = student.SubjectId;
            _context.SaveChanges();
            return RedirectToAction(nameof(Index));
        }
```

If you want to put Validation that <u>Email Id</u> is not duplicate:-

- Go to StudentController file.

- In case of Create

```
[HttpPost]
     public ActionResult Create(Student student)
        {
          if (student == null)
              return HttpNotFound();
          //Validation Server Side
          if (!ModelState.IsValid)
          {
              return View();
          }
          //Duplicate Email Validation
var duplicate = _context.students.FirstOrDefault(s => s.Email ==
student.Email);
            if(duplicate!=null)
            {
                ModelState.AddModelError("Email", "Email in Use");
                return View();
            }
            _context.students.Add(student);
            _context.SaveChanges();
            return RedirectToAction("Index");
        }
```

- In case of Edit.

```
[HttpPost]
        public ActionResult Edit(Student student)
        {
            if (student == null)
                return HttpNotFound();
            var studentFromDb = _context.students.Find(student.Id);
            if (studentFromDb == null)
                return HttpNotFound();
            if (!ModelState.IsValid)
            {
                ViewBag.SubjectList = _context.subjects.ToList();
                return View(studentFromDb);
            }
            //Unique Email Validation
var duplicate = _context.students.FirstOrDefault(s => s.Email ==
student.Email);
```

```
var emailcheck = _context.students.FirstOrDefault(s => s.Id ==
student.Id);
            if (duplicate != null)
            {
          if (!(duplicate.Id == emailcheck.Id && duplicate.Email ==
emailcheck.Email))
                {

                 ModelState.AddModelError("Email", "Email in Use");
                    return View();
                }
            }
            studentFromDb.Name = student.Name;
            studentFromDb.Age = student.Age;
            studentFromDb.Email = student.Email;
            studentFromDb.SubjectId = student.SubjectId;
            _context.SaveChanges();
            return RedirectToAction(nameof(Index));
        }
```

Now run it.

Custom Validation:-

Create new folder:-

- Right click on project.
- Add new folder.
- Set name as "**Custom**".
- Now add a **Class**.
- Set name as "**CustomAgeValidatiion**".
- Click on add button.

```
public class CustomAgeValidation:ValidationAttribute
    {
        protected override ValidationResult IsValid(object value,
ValidationContext validationContext)
        {
            var age = Convert.ToInt32(value);
            if (age >= 18 && age<=80)
                return ValidationResult.Success;
            else
                return new ValidationResult("Age must be 18-80");
        }
    }
```

- Now go to Student Model.
- Go to Age line and Paste "`CustomAgeValidation`" in [] braces.

```
[Required]
        //[Range(18, 40, ErrorMessage = "Enter Valid Age")]
        [Display(Name = "Student Age")]
        [CustomAgeValidation]
        public int Age { get; set; }
```

- Now create a class in **Custom** Folder.
- Set name as "CustomEmailValidation".
- Click on **Add** Button.

```
public class CustomEmailValidation:ValidationAttribute
    {
        protected override ValidationResult IsValid(object value,
ValidationContext validationContext)
        {
            var email = value.ToString();
            if(Regex.IsMatch(email,@"^[^@\s]+@[^@\s]+\.[^@\s]+$"))
            {
                return ValidationResult.Success;
            }
            else
            {
                return new ValidationResult("Enter Valid Email");
            }
        }
    }
```

Go to Browser and search 'Regex Email in C#'.

- Open **Student** Model.
- Go to Email Line and paste "`CustomEmailValidation`" in [] braces.
- Now Run it.

How to Create Layout:-

- Go to **View**.
- Then **Shared** folder.
- Copy the **_Layout.cshtml** file and paste it on same route.
- Rename this file "**_MyLayout.cshtml**".
- Download **Bootstrap Theme** & apply on it.
- Go to **App_Start**.
- Then open **BundleConfig.cs** file.
- Select and copy "`bundles.Add(new StyleBundle("~/Content/css").Include("~/Content/bootstrap.css","~/Content/site.css"));`" this syntax and paste it on same location.
- Now paste the downloaded theme name in it.
- Go to **_MyLayout** file.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    @Styles.Render("~/MyContent/css")
```

- Now go to **View** folder in **Solution Explorer**.
- Then **Student** folder.
- Open **Index.cshtml** file.

```
@model IEnumerable<Code_First_Approach.Models.NewStudent>
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

This method is used for specific Action.

For Apply in all Actions:-

- Select & Copy **_ViewStart.cshtml** file from **Views** folder.
- Paste it in **Student**(View's folder).
- Now open this file.

Change its name with Your file name(_MyLayout).

```
@{
    Layout = "~/Views/Shared/_MyLayout.cshtml";
}
```

Day31

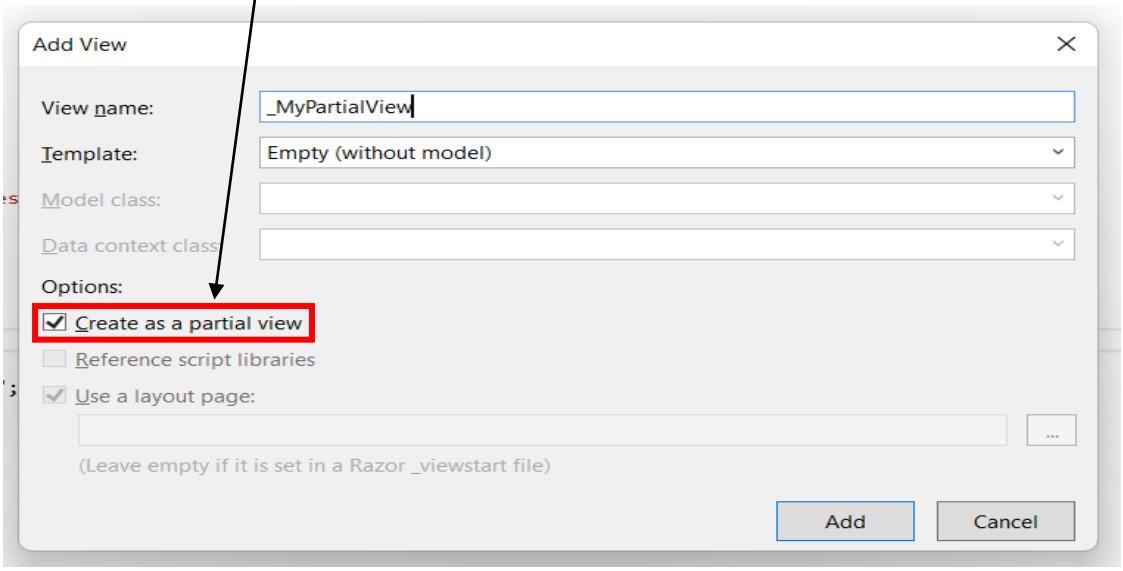
- Go to **Controller** folder in **Solution Explorer**.
- Then open **HomeController**.
- Now go to **Contact** Action.

```
public ActionResult Contact()
{
    ViewBag.Message = "Your contact page.";
    return View("Contact", "_MyLayout");
}
```

Type these Syntax.

How to Create Partial View

- Go to **Views** folder.
- Then **Shared** folder.
- **Add a View** in **Shared** folder.
- Set name as '**_MyPartialView**'.
- Click on **Partial View** Checkbox.



- Click on **Add** button.
- Now go to **_Layout** file.

- Select & Copy the Header Navigation.

```
<div class="navbar navbar-inverse navbar-fixed-top">
  <div class="container">
     <div class="navbar-header">
         <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
             <span class="icon-bar"></span>
             <span class="icon-bar"></span>
             <span class="icon-bar"></span>
         </button>
         @Html.ActionLink("Application name", "Index", "Home", new
{ area = "" }, new { @class = "navbar-brand" })
     </div>
     <div class="navbar-collapse collapse">
         <ul class="nav navbar-nav">
             <li>@Html.ActionLink("Home", "Index", "Home")</li>
             <li>@Html.ActionLink("About", "About", "Home")</li>
             <li>@Html.ActionLink("Contact", "Contact",
"Home")</li>
             <li>@Html.ActionLink("Student List", "Index",
"Student1")</li>
         </ul>
     </div>
  </div>
</div>
```

- And paste in **_MyPartialView** file.
- Now call the **_MyPartialView** file.
- To call this file open **_Layout** file and then

```
<body>
@Html.Partial("_MyPartialView")
    <div class="container body-content">
```

Or

```
<body>
    @{
        Html.RenderPartial("_MyPartialView");
    }
<div class="container body-content">
```

Or

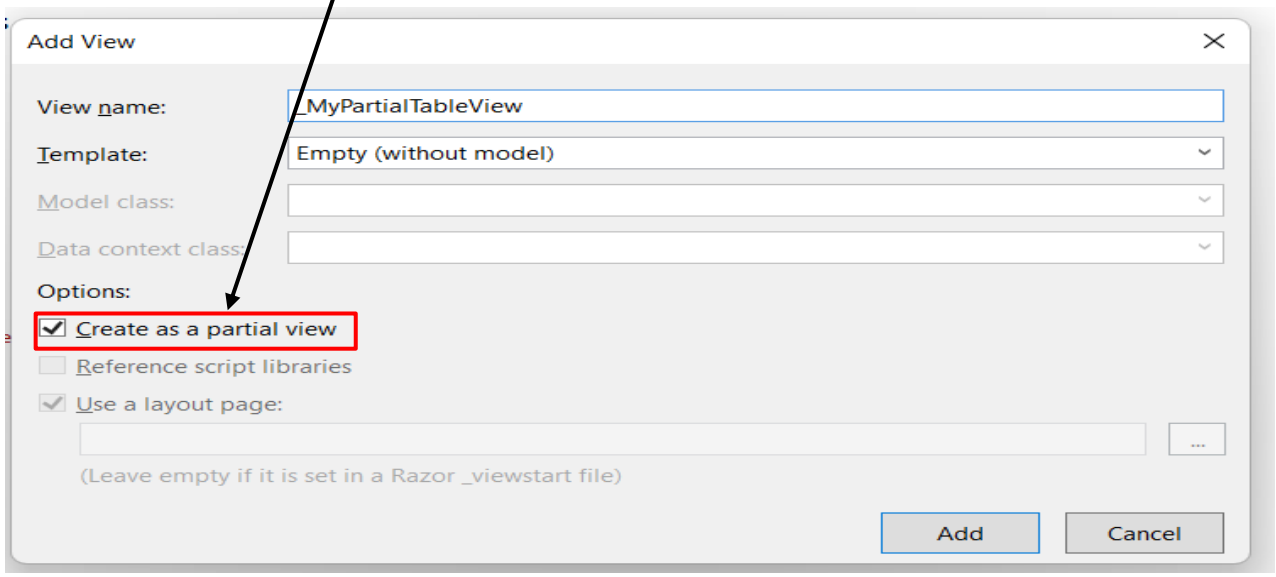
```
<body>
    @Html.Action("LoadAction")
<div class="container body-content">
```

- Now go to **Controller** folder.
- Go to **HomeController**.
- Create an **Action**.

```
public ActionResult LoadAction()
        {
            return PartialView("_MyPartialView");
        }
```

Create another partial View

- Go to **Solution Explorer**.
- Go to **Views** folder.
- Then **Shared** folder.
- Add a **View** by right clicking on **Shared** folder.
- Set name as "**_MyPartialTableView**".
- Click on **Partial View** Checkbox.



- Click on Add Button.
- Now do coding in Current file.

```
<table class="table-condensed table-bordered">
    <tr>
        <th>Name</th>
        <th>Age</th>
    </tr>
    <tr>
        <th>Jarvis Stark</th>
        <th>22</th>
    </tr>
    <tr>
        <th>Amit Kumar</th>
        <th>22</th>
    </tr>
</table>
```

- Now go to **Views** folder in **Solution Explorer**.
- Then **Home** folder.
- After that open "**About.cshtml**" file.

```
@{
    ViewBag.Title = "About";
}
<h2>@ViewBag.Title.</h2>
<h3>@ViewBag.Message</h3>
@Html.Partial("_MyPartialTableView")

<p>Use this area to provide additional information.</p>
```

This Syntax is Used to Call _MyPartialTableView

Data Storage

Types of Data Storage:-

1. VIEWDATA
2. VIEWBAG
3. TEMPDATA
4. SESSION

Note:- We can store any type of Data in ViewData & ViewBag.
Scope of ViewData & ViewBag is **Controller** to **View** only.

VIEWDATA	VIEWBAG
TYPE CASTING IS REQUIRED.	Type casting is not required.
IT DOESN'T MAKE DYNAMIC PROPERTIES.	It makes Dynamic properties.

- Now go to **StudentController**.
- Go to **Index Action**.

Request Controller to View:-

```
public ActionResult Index()
        {
            ViewData["myViewData"] = "StudentList";
            var studentindb = _context.students.Include(s =>
s.subject).ToList();
            return View(studentindb);
        }
```

- Go to **View** of **Index** in **StudentController**.

```
<h1>@ViewData["myViewData"]</h1>
@Html.ActionLink("Call Next Page","Contact","Home")
```

- Go to **Index Action** in **StudentController**.

```
public ActionResult Index()
        {
            ViewData["myViewData"] = "StudentList";
            var studentindb = _context.students.ToList();
            return View(studentindb);
        }
```

Day32

- Go to HomeController.
- Create New Action.

```
public ActionResult DemoActionFirst()
        {
            ViewData["vData"] = "MyViewData";
            return RedirectToAction("Index", "Student");
        }
```

- Create its View.
- Select ViewData then right click on it then **Add Watch** during execution of program.
- Go to **StudentController**.
- Then **Index** Action.

```
public ActionResult Index()
        {
            var data = ViewData["vData"];
            ViewData["myViewData"] = "StudentList";
            //var studentindb = _context.students.ToList();
            var studentindb = _context.students.Include(s =>
s.subject).ToList();
            return View(studentindb);
        }
```

- Go to **HomeController**.
- Create an Action "**DemoActionSec**".

```
public ActionResult DemoActionSec()
        {
            var data = ViewData["vData"];
            return View();
        }
```

- Add a **Model**.
- Set name as "**Subject**".

```
public class Subject
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
```

- Go to "**ApplicationDbContext**" file.

```
public DbSet<Subject> subjects { get; set; }
```

- Now go to **Tools** tab.
- Then **NuGet package Manager** option.
- Now click on **NuGet package Console**.
- Now **add-migration AddSubjectTable**.
- Then **update-database**.

Create a Foreign Key

- Go to **Student** Model.

```
public class Student
    {
        public int Id { get; set; }
          ¦
          ¦
          ¦
          ¦
        public int SubjectId { get; set; }
        public Subject subject { get; set; }
    }
```

- Now go to **Tools** tab.
- Then **NuGet package Manager** option.
- Now click on **NuGet package Console**.
- Now **add-migration AddForeignKeyToSubjectId**.
- Then **update-database**.

- Again **add-migration emptyMigrate**.

```
public partial class emptyMigrate : DbMigration
    {
        public override void Up()
        {
            Sql("insert subjects values('English')");
            Sql("insert subjects values('Maths')");
            Sql("insert subjects values('Hindi')");
            Sql("insert subjects values('Punjabi')");
        }

        public override void Down()
        {
        }
    }
```

- Now **update-database**.
- Open **StudentController**.
- Go to **Create Action**.

```
public ActionResult Create()
        {
            ViewData["SubjectList"] = _context.subjects.ToList();
            return View();
        }
```

- Open **Create.cshtml** file.

```
<div class="form-group">
        @Html.LabelFor(m => m.SubjectId)
        @Html.DropDownListFor(m=>m.SubjectId, new
SelectList(ViewData["SubjectList"] as
List<Subject>,"Id","Name"),"Select Subject", new {@class="form-
control"})
</div>
```

Now Add a NameSpace to call this Class.

```
@using Code_First_Approach.Models
```

NameSpace of Subject Class.

- Open **StudentController**.

- Now go to **Create Action**.

```
public ActionResult Create()
        {
            ViewBag.SubjectList = _context.subjects.ToList();
            return View();
        }
```

- Open **Create.cshtml** file.

```
    <div class="form-group">
        @Html.LabelFor(m => m.SubjectId)
        @Html.DropDownListFor(m => m.SubjectId, new
SelectList(ViewBag.SubjectList, "Id", "Name"), "select subject",
new { @class = "form-control" })
    </div>
```

How to Show Subject name in Student DataBase:-

- Add a namespace in **StudentController**.
- Now go to **Index Action**.

```
public ActionResult Index()
        {
            var studentindb = _context.students.Include(s =>
s.subject).ToList();
            return View(studentindb);
        }
```

- Now go to **Index.cshtml** file.

```
<tr>
  |
  |
  |
  |
    <td>@student.subject.Name</td>
  |
  |
  |
  |
</tr>
```

To show DropDown List in Edit:-

- Go to **Edit Action**.

```
public ActionResult Edit(int id)
        {
            if (id == 0)
                return HttpNotFound();
            var studentInDb = _context.students.Find(id);
            if (studentInDb == null)
                return HttpNotFound();
            ViewBag.SubjectList = _context.subjects.ToList();
            return View(studentInDb);
        }
```

- Go to **Edit.cshtml** file.
- Then copy the dropdown syntax from "**Create.cshtml**" file and paste in "**Edit.cshtml**" file.
- Now go to "**POST Edit Action**".

```
public ActionResult Edit(Student student)
  {
      if (student == null)
          return HttpNotFound();
      var studentFromDb = _context.students.Find(student.Id);
      if (studentFromDb == null)
          return HttpNotFound();
      if (!ModelState.IsValid)
      {
          ViewBag.SubjectList = _context.subjects.ToList();
          return View(studentFromDb);
      }
      //Unique Email Validation
      var duplicate = _context.students.FirstOrDefault(s => s.Email
== student.Email);
      var emailcheck = _context.students.FirstOrDefault(s => s.Id
== student.Id);
      if (duplicate != null)
      {
          if (!(duplicate.Id == emailcheck.Id && duplicate.Email ==
emailcheck.Email))
          {
              ModelState.AddModelError("Email", "Email in Use");
              return View();
          }
      }
      studentFromDb.Name = student.Name;
      studentFromDb.Age = student.Age;
      studentFromDb.Email = student.Email;
      studentFromDb.SubjectId = student.SubjectId;
      _context.SaveChanges();
      return RedirectToAction(nameof(Index));
  }
```

Now run it.

Create New Project(UPSERT)

- Open **Visual Studio Application**.
- Click on **Create New Project**.
- Set name as "**WebApllication_Employee_Dep_Deg_Upsert**".
- Now apply a Theme.
- Add connection in **web.config** file.
- Install **EntityFramework** package.
- Create Model & set name as "**Department**".
- Add its properties(**Id, Name**).
- Add another Model named with "**Designation**".
- Add its properties(**Id, Name**).
- Add another Model named with "**Employee**".
- Add its properties(**Id, Name, Address, Salary**).
- Add a **Foreign Key** after **Salary property**.

```
//for Department
        [Display(Name="Department")]
        public int DepartmentId { get; set; }
        public Department department { get; set; }
//for Designation
        [Display(Name="Designation")]
        public int DesignationId { get; set; }
        public Designation designation { get; set; }
```

- Now create a folder named with "**Data**".
- Now create a class in **Data** folder.
- Set name as "**ApplicationDbContext**".
- Add all three properties in current file(**Department, Designation, Employee**).

```
public class ApplicationDbContext:DbContext
    {
        public ApplicationDbContext():base("constr")
        {

        }
        public DbSet<Department> departments { get; set; }
        public DbSet<Designation> designations { get; set; }
        public DbSet<Employee> employees { get; set; }
    }
```

- Now go to **Tools** tab in menu bar.
- Then choose **NuGet Package Manager** option.
- After that click on **Package Manager Console**.
- Now enable-migrations.
- add-migration initload.
- update-database.

Day34

How to enter Data in Department & Designation Table

- Add empty migration.

add-migration populateDepDsg

```
public override void Up()
        {
            //dep
            Sql("insert departments values('Sales')");
            Sql("insert departments values('Marketing')");
            Sql("insert departments values('Computer')");
            //dsg
            Sql("insert designations values('Project Manager')");
            Sql("insert designations values('Team Leader')");
            Sql("insert designations values('Programmer')");
        }
```

- **update-database**.
- Create a Controller named with "**EmployeeController**".

```
public class EmployeeController : Controller
    {
        private readonly ApplicationDbContext _context;
        public EmployeeController()
        {
            _context = new ApplicationDbContext();
        }
        protected override void Dispose(bool disposing)
        {
            _context.Dispose();
        }
        // GET: Employee
```

- Now create an Action “**Index**”.

```
public ActionResult Index()
        {
            var emplist = _context.employees.Include(s =>
s.department).Include(s => s.designation).ToList();
            return View(emplist);
        }
```

- Now add a view named with “**Index**”.

```
@model
IEnumerable<WebApplication1_Employee_Dep_Dsg_upsert.Models.Employee
>
@{
    ViewBag.Title = "Index";
}

<h2 class="text-info text-center">Employee List</h2>
@Html.ActionLink("Create New", "Upsert", new {@class="btn btn-
link"})
@if(!Model.Any())
 {
  <p class="alert-danger text-center" style="border-block:double">
No Data Found</p>
 }
else
     {
       <table class="table table-bordered table-responsive">
           <thead style="background-color:red">
               <tr>
                   <th>Name</th>
                   <th>Address</th>
                   <th>Salary</th>
                   <th>Department</th>
                   <th>Designation</th>
                   <th>Actions</th>
               </tr>
           </thead>
<tbody style="background-color:blanchedalmond">
 @foreach(var item in Model)
  {
    using (Html.BeginForm("Delete", "Employee", new {@id=item.Id}))
     {
```

```
        <tr>
         <td>@item.Name</td>
         <td>@item.Address</td>
         <td>@item.Salary</td>
         <td>@item.department.name</td>
         <td>@item.designation.Name</td>
         <td>
        @Html.ActionLink("Edit", "Upsert", new { id = item.Id })|
        @Html.ActionLink("Detail", "Details", new { id = item.Id })|
        <input type="submit" value="Delete" onclick="return
confirm('wanna delete this file')" class="btn btn-success" />
         </td>
        </tr>
       }
          }
            </tbody>
        </table>
      }
```

- Create an **Action** named “**Upsert**”.

```
public ActionResult Upsert(int? id)
    {
        ViewBag.departmentlist = _context.departments.ToList();
        ViewBag.designationlist = _context.designations.ToList();
        Employee employee = new Employee();
        if(id==null)
        {
            return View(employee);
        }
        else
        {
            var empfromdb = _context.employees.Find(id);
            return View(empfromdb);
        }
    }
```

Day35

- Now add **View** named with "**Upsert**".

```
@model WebApplication1_Employee_Dep_Dsg_upsert.Models.Employee
@{
    ViewBag.Title = "Upsert";
    var title = Model.Id == 0 ? "New Employee" : "Edit Employee";
}

<h2 class="text-uppercase">@title</h2>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    @Html.ValidationSummary(false, "", new { @class = "text-danger"
})
    <div class="form-group row">
        <div class="col-lg-4">
            @Html.LabelFor(m => m.Name)
        </div>
        <div class="col-lg-8">
            @Html.TextBoxFor(m => m.Name, new { @class = "form-
control" })
        </div>
    </div>
    <div class="form-group row">
        <div class="col-lg-4">
            @Html.LabelFor(m => m.Address)
        </div>
        <div class="col-lg-8">
            @Html.TextBoxFor(m => m.Address, new { @class = "form-
control" })
        </div>
    </div>
    <div class="form-group row">
        <div class="col-lg-4">
            @Html.LabelFor(m => m.Salary)
        </div>
        <div class="col-lg-8">
            @Html.TextBoxFor(m => m.Salary, new { @class = "form-
control" })
        </div>
    </div>
    <div class="form-group row">
        <div class="col-lg-4">
```

```
            @Html.LabelFor(m => m.DepartmentId)
        </div>
        <div class="col-lg-8">
            @Html.DropDownListFor(m => m.DepartmentId, new
SelectList(ViewBag.departmentlist, "Id", "Name"), "Select
Department", new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group row">
        <div class="col-lg-4">
            @Html.LabelFor(m => m.DesignationId)
        </div>
        <div class="col-lg-8">
            @Html.DropDownListFor(m => m.DesignationId, new
SelectList(ViewBag.designationlist, "Id", "Name"), "Select
Designation", new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group row">
        <div class="col-lg-4">
            @if (Model.Id == 0)
            {
                <input value="Save" type="submit" class="btn btn-
success" />
            }
            else
            {
                <input value="Update" type="submit" class="btn btn-
success" />
            }
        </div>
        @Html.ActionLink("Back to List","Index")
    </div>
}
@section scripts
{
 @Scripts.Render("~/bundles/jqueryval")
}
```

- Now go to **Controller**.

```
[HttpPost]
[ValidateAntiForgeryToken]
  public ActionResult Upsert(Employee employee)
    {
      if (employee == null)
        return HttpNotFound();
      if(!ModelState.IsValid)
        {
          ViewBag.departmentlist = _context.departments.ToList();
          ViewBag.designationlist = _context.designations.ToList();
          return View(employee);
        }
      if(employee.Id==0)
       {
         _context.employees.Add(employee);
       }
      else
       {
          var employeeindb = _context.employees.Include(s =>
s.department).Include(s => s.designation).FirstOrDefault(s => s.Id
== employee.Id);
       if (employeeindb == null)
         return HttpNotFound();
         employeeindb.Name = employee.Name;
         employeeindb.Address = employee.Address;
         employeeindb.Salary = employee.Salary;
         employeeindb.DepartmentId = employee.DepartmentId;
         employeeindb.DesignationId = employee.DesignationId;
       }
        _context.SaveChanges();
        return RedirectToAction("Index");
    }
```

How to change Button Name with condition during Execution of Project:-

- Go to **Upsert.cshtml** file.
- Go in Last.

```
<div class="form-group row">
  <div class="col-lg-4">
   @if (Model.Id == 0)
   {
     <input value="Save" type="submit" class="btn btn-success" />
   }
   else
   {
     <input value="Update" type="submit" class="btn btn-success" />
   }
  </div>
   @Html.ActionLink("Back to List","Index")
</div>
```

- Now create an **Action**.
- Set name as "**Details**".

```
public ActionResult Details(Employee employee)
        {
            if (employee == null)
                return HttpNotFound();
            var emp =
_context.employees.Include(s=>s.department).Include(s=>s.designatio
n).FirstOrDefault(s=>s.Id==employee.Id);
            if (emp == null)
                return HttpNotFound();
            return View(emp);
        }
```

- Now create its View.

```
@model WebApplication1_Employee_Dep_Dsg_upsert.Models.Employee
@{
    ViewBag.Title = "Details";
}

<h2>Employee Details</h2>
<table class="table table-bordered table-responsive">
    <thead>
        <tr>
            <th>Name</th>
            <th>Address</th>
            <th>Salary</th>
            <th>Department</th>
            <th>Designation</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>@Model.Name</td>
            <td>@Model.Address</td>
            <td>@Model.Salary</td>
            <td>@Model.department.name</td>
            <td>@Model.designation.Name</td>
        </tr>
    </tbody>
</table>
<div>
```