⊙ Dependency Injection/ Inversion of Control (IoC)

→ Application is divided into 3 parts/layer dependant on each other.

→ If one change other also change.

→ So we do loose coupling to remove dependency. Plug and Play environment

→ DI / IoC is structure used for design loosely coupled system.

E.g:

```
class FileWriter{

    public void save(string data)
    {
        =
    }
}
```

→ class Program

```
{   FileWriter fw = new FileWriter();
    fw.Save(_);
}
```

: Now we wanted to save in Database.

→ class SqlWriter

```
{   public void Save(_)
    [   =
    }
```

: There is fw dependency
: So we make interface

* Class represents real-world objects (Properties and functions of it).

Interfacing:

→ What the class will do, it is defined in interface. Not How part define.
(contract)

→ Class for interface and objects/class

→ No Implementation of object is defined here.

→ Abstract Class (No attributes)

→ Interface Class always start with capital "I".

```
interface IWriter{                    : Not clear where to save
    public void Save(string data);
}
-> class SqlWriter : IWriter {
    public void Save ( .. )
    {
    }
}
                                  class FileWriter:IWriter{
                                  }
```

class Program

{

IWriter fw = new FileWriter(...);

fw.Save(....);

}

still object Made.

We use IoC

We will make class object outside and inject it to Program code.

(Dependency Injection)

class

→ public Program {

private IWriter _Iw

public Program(IWriter Iw)

{

_IW = Iw

}

_Iw.save(...);

}

⊙ In MVC:

→ Controller Layer get data from Repositories.

Controllers	Repositories
Repository rp=new ...(); rp.save();	↳ Product Register { public void Save() { } }

→ public class Program

{ private IRegister = _Ir;

}

public Program(IProductRegister Ir)

{

}

⊙ Model → class → Customer.

public class Customer

{

int Id

string Name

}

→ Model → Interface → New Item

(Folder)

ICustomer ← Interface

public interface ICustomer

{

public void save (customer c)

Upalate

Delete

}

→ New Folder → Repository

CustometerRepository

: Implement Interface (Option

: public class CustomerRepository : ICustomer

{

Save
Update
Delete
GetAll

}

CustomerController

{ private readonly ICustomer _customer;

public Customer(I Customer c)

{ _customer = c;

}

public IActionResult Index()

{ return (_customer.GetAll());

}

→ GetAll in Repository.

public List<Customers> GetAll()

{ List<Customers> L = new List....;

List.Add(new Customer { Id=1, Name="1"

"

"

2 C2

3 C3

return (L); 4 C4

-> View -> Customer Folder -> Index

@model IEnumerable<Customer>
List<Customer>

```
<table>
  @foreach(var c in Model)
  {
    <tr> <td>@c.Id</td>
         <td>@c.Name</td>
    </tr>
  }
</table>
```

-> Program.cs

-> builder.Services.AddTransient<ICustomer, CustomerRepository>();

(In Main(string[] args))

: Practice Dependency Injection

: Quiz on Wednesday (BootStrap)

3/12/24

⊙ Dockerization :

Close all the dependencies and environment of application in container and ship it.

-> Used for containerization and make deployment easy.

-> Firstly use it for console App and use it on command prompt.

-> Till Next Friday Remaining Front-end.
(Also Dockerize it and provide Docker image).
(Docker Hub Registry Link) (13th December)
(i) 100% Frontend
(ii) Dockerize Frontend

: Theme Wagon

⊙ CSS Positioning:

→ Five Types of Positioning:

(i) Static (Default) (ii) Relative

(iii) Absolute (iv) Fixed

(v) sticky

→ VS-code → css_position_example.html

→ <body>

" " "two"

" " "three"

→ <head>

<style>

#parent {

border: 1px solid red;

padding: 20px;

width: 50%;

height: 400px;

}

#one

{ width: 100px;

height: " ; display: inline / block;

: same for two, three

background-color-red

⊙ relative: (i) move with respect to it
(ii) Not leaves his space/position

→ relative #two {

" "

position : relative

top: 250px

left: 500px;

}

→ absolute

#two {

position : absolute

top: 0px

left: 0px

}

: (i) calculates it position from parent
(ii) Leaves his space/ position

parent { " position: relative

}

: Now two will be according to positional parent.

→ fixed #two { "

position: fixed;

bottom: 100px;

left: 100px

}

: Fixed position is given from bottom.
: It remains at specific place on scroll.

→ sticky #four {

(with respect to its container stick)

position: sticky

top: 0px;

}

⊙ Float:

(used for alignment)
(to overcome regular flow)

 class container
<img> ....
<p> ....

```
image{
    float: left
}
```

: elements comes out of normal behaviour
: Add with next line (side by side)

⊙ clear:

(All float element should remove that behavior and move to next line)

→ both, left, right null

: Adding Folder or picture

→ <header id="header">
<h1> My Blog </h1>

5/12/24

⊙ File Uploading:

→ Home Controller.

index.cshtml:

<form method="post" enctype="multipart/form-data"

asp-action="Index" asp-controller="Home">

 Select File:

<input type="file" name="postedFiles

<button type="submit">Upload File

</button>

</form>

→ wwwroot → Upload Folder creation on Runtime

→ Index Method:

[HttpGet] → Request

Post

Index (List<IFormFile> postedFiles)

{

string wwwPath = this

: Next Page

: private readonly IWebHostEnvironment

Environment;

Public HomeController(IWebHostEnvironment e)

{ Environment = e

}

post

```
① Index( List<IFormFiles> postedFiles)
  {
    string wwwPath = this.Environment.
                     WebRootPath;
    string path = Path.Combine(wwwPath,
                  "Uploads");
    if (!Directory.Exists(path))
    {
      Directory.CreateDirectory(path);
    }

    foreach (var file in postedFiles)
    {
      var fileName = Path.GetFileName
                     (File.FileName)
      var pathwithFileName = Path.
                     Combine(path,
                     fileName)
      using(FileStream stream = new
            FileStream(pathwithFileName,
            FileMode.Create))
      {
        file.CopyTo(stream);
        ViewBag.Message = "File Uploaded
                          Successfully";
      }
    return View();
  }
```

This path in DB

It automatically closes

: In Database (File path attribute) Lecture 64

⊙ Media Queries:

→ According to dimension CSS also changes

→ We define different media queries for different CSS.

→ General HTML:

Then CSS:

```
header {
        }
        :
```

→ @media only screen and (min-width : 600px)

```
{ header {
        padding: 30px;
        background-color: aqua;
        }
}
```

→ @media only screen and (min-width 900px)

```
{
  header {
        }
}
```

⊙ Flex-Box:

→ Make Responsive without position and queries.

```
<body>
  <div class="flex-container">
    class="flex-child"
    <div> 1 .—
    <div> 2 —
    <div> 3 —
    <div> 4 . .
  </div>
</body>
```

: By default it is horizontal

: <head>

: <style>

.Flex container

{display: flex

}

.flex-child

{ background-color: wheat

margin: 20px

padding: 20px

}

: Adding more elements to observe effect (1-12)

⊙ Properties:

(i) flex-direction : column (row) (row-reverse) (column-reverse)

(ii) flex-wrap : nowrap (Adjust in single line)

: wrap (Next-Line)

: wrap-reverse (opposite-element)

(iii) Justify content: center (Horizontal alignment)

: flex-start

: flex-end

: space-around

: space-evenly

: space betwe

(ii) text-align : center.

OR

align items: center : flex-end
(vertical alignment) : flex-start : stretch

-> align-content : space-between

(all of justify content effects)

: height: 400px (flex-container)

: containers properties

*

child properties

" " two " "

" " three " "

<style>

one {

order: 2

: flex-grow: 1

or

flex shrink: 3

: Increases with respect to other

two {

order: 1

flex-grow 9

* Practice Flex-Box Example.

: object oriented
: Relation

⑭ ORM: (Object Relation Mapper)

→ If we are only familiar to OOP and not familiar with SQL, then we use ORM.

→ ORM are used in every language to manage database without using queries

→ Types:

(i) Dapper (Make Performance Better) (Automatically return object) (Using SQL queries and returns data in object form)

(ii) Entity Framework core (Do every work in object form)

→ Mapper is intermediate layer between object oriented and ~~with~~ queries. (It translates object and queries)

→ Better is ADO.NET. (Direct Method).

→ Performance matter (Dapper) otherwise Entity Framework core.

→ Entity Framework Core:

Two Approaches:

(i) Database First (If already database present create classes)

(ii) Code First (classes defined, then database will be created with EF core)

⊙ Code First:

→ We create: (i) Model Classes against Entities.
(ii) DB Context Class (Bridge work)

→ Then we make database and do migration (db translation, eventually map it to DB) (code generated to update)

→ If we add properties in model class (suppose cgpa), then it would be reflected in database.

→ Model Classes → DB context class → Migration

→ Implementation:

→ Console App → Add new class → Student

```
: internal class Student
{   public int id { get; set; }
    "  string Name  "
    "  string RollNO. "
}
```

→ Add → New class → My Application context.

: Classes and DBSet

Student (DB required) , Courses (DB required) Student Course (No DB required)

internal class MyApplicationContext : DBContext

: Right Click and select core options.

: Tools → Browse → Entity Framework (First and other also)

(Microsoft.EntityFrameworkCore.SqlServer)

(Install)

(using Microsoft Services)

: using Microsoft.EntityFrameworkCore

public DBSet<Student> Students {get;set}

→ protected override void OnConfiguring (DBContextOptionsBuilder optionBuilder)

{ optionBuilder.UseSqlServer("connection string");

: change catalogue = MystWeb

→ Now Migration:

⊖ Classes would be required to download and run migration on cmd.

⊙ Entity Framework on Microsoft

Getting started ← Your First EF Core App

Create Database → Third Command

① dotnet ef migrations Add InitialCreate

→ Tools → Browse → Entity Framework and (Install) Design

→ _Initial Create, Up and Down Method (Students)

→ copy ④ dotnet update (From website)

→ Then check for Table creation in mySEWeb Database and table student.

→ Student Class new properties:

.....

public float CGPA { get; set; }

: Do Migration:

dotnet of migrations AddCGPAinColumn

dotnet EF update.

→ For New Class:

→ Class → DBSet → Migration

: Class Department

{ public int Id { get; set; } : By Default Primary

public String Name { get; set; }

}

```
[Key]
public int deptId{get; set;}
```

→ contextClass:

```
{ "..."
    public DbSet<Departments>
                Departments{get; set;}
}
```

→ Migration:

```
dotnet Add NewTableCreation
dotnet EF update.
```

... Check for migration folder (up, down)

⊙ Code First Advantage (Stale Maintain of DB Evolve).

12/12/24

⊙ Insertion using EF:

```
=> class Program
   {
      static void Main(....)
      { Student student= new Student();
        student.Name="Ali"
        student.RollNumber="1"
        student.CGPA=4.0f
```

: DBSet
In Memory Table

```
using (MyApplicationContext context =
                    new MyApplicationContext())
{
    context.Students.Add(student);
    context.SaveChanges();
}
```

→ For Department:

```
Main(...)
{ Department d = new Department();
  d.Id = 1;
  d.Name = "Software Engineering";
  using (MyApplicatio context =
                    new MyApplicationContext())
  { context.Student/Departments.Add(d);
    context.SaveChanges();
  }
}
```

→ All students from table:

→ Reading -- LINQ (Language Integrated Query.

In Below using ---

```
var data = context.Students.ToList();
foreach (var item in data)
{
    Console.WriteLine(item.Name);
}
```

⊙ //print all departments

```
var data = context.Departments.ToList();
foreach (var data1 in data)
{
    Console.WriteLine(data1.Name);
}
```

⊙ Where CGPA is —:

```
var data = context.Students.Where(S =>
                S.CGPA >= 3
                .ToList();
```

⊙ Only RollNo or Name Required:

```
var data = context.Students.Where(S => S.CGPA >= 3)
            .Select(a => a.RollNumber)
            .ToList();
```

→ Select both Name & RollNo.

```
//          //          //
            .Select(a => new { N = a.Name,
                               R = a.RollNumber })
                .ToList()
```

⊙ CGPA > 3 & Name = 'S' / 'O'

and return Name, Roll, CGPA

```
//  //
var data = context.Students.Where
                (S => S.CGPA > 3 ||
                S.Name.StartWith('S')   S.Name Like 'S%').
```

N = a.Name, C = a.CGPA }

Correct → " " context.Students.
Where(s => s.Name.StartsWith('s')),
Where(s => sCGPA > 3).Select(a =>
new { N = a.Name, R = a.RollNumber,
C = a.CGPA })

→ Update Data:

→ First we get data and assign new values and call save changes.

(returning 1st object)

```
var x = context.Students.First();
x.CGPA = 1f;
context.SaveChanges();
```

First(s => S.Name == "Tom")

→ Delete Data:

→ Get Data First:

```
var y = context.Students.First();
context.Students.Remove(y);
context.SaveChanges();
```

: Can use where condition : then edit or remove by using loop

: var y = context.Students.Single(s => S.Name == 'Ali')

: Single (If more than one exists it throw exception)

: First (If more than one exists, it returns first only).

: We can call savechanges() only

→ Student Belongs to Department

: Class Student

```
{
    ...

    ...
    public Department
                Department {get; set;}
}
```

: Department
many ↓↑ one
Student
(Many to one)

: class Department

```
{
    ...

    public List<students> Students
                    {get; set;}
}
```

: Now run Migrations :

dotnet ef migrations add AddColumn

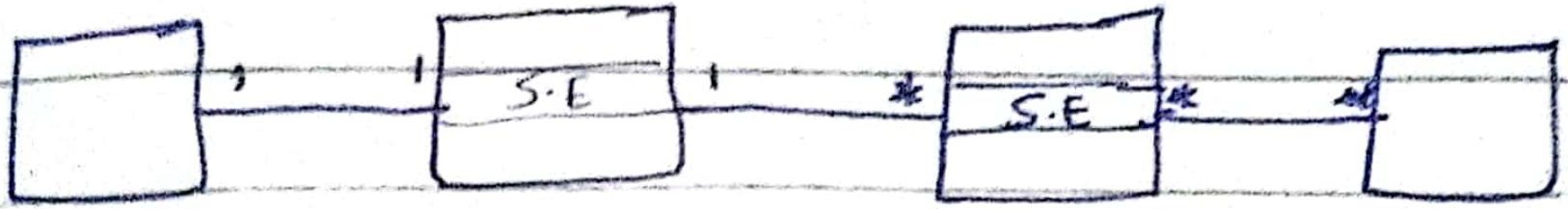
⊙ Indexing :

→ Record of Data (Book Example
Index : Page Number)

→ In DB Index is taking space but make our work easy.

→ cascading delete both data in local and foreign Key

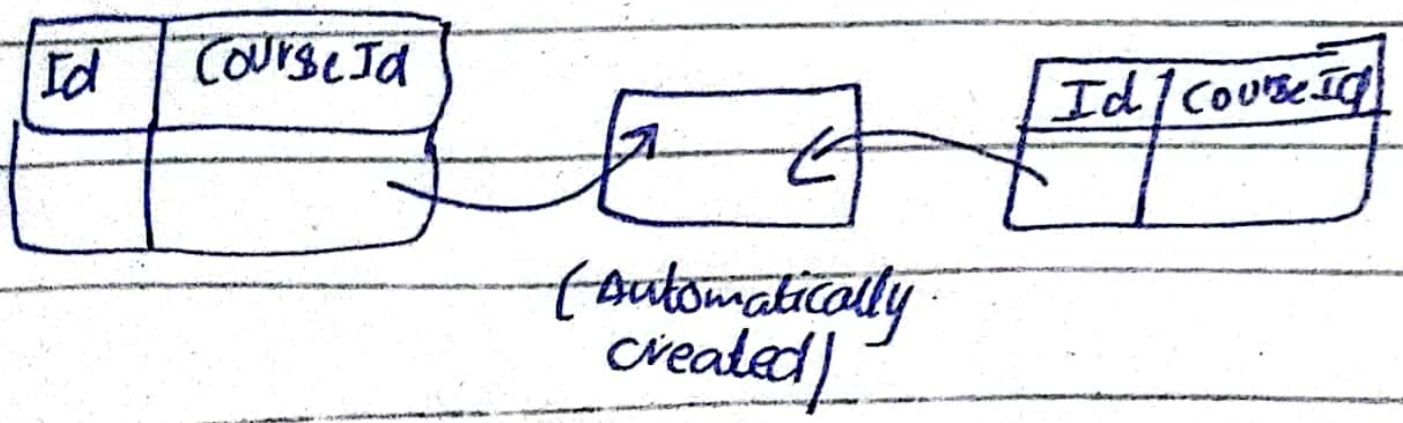
: onDelete : ReferentialAction.Cascade.



DeptDetail | Dept | Studant | Course

F.K(Dept) | F.K(DeptDetail) | List<Courses> | List<Studeats>

⊙ If relations is many to many.



⊙ User - Management:

→ Authenticate (Validate User)

(Wrong Input → NO Access).

: 3 ways of web application
→ Razor Pages
→ MVC
→ Blazor

→ Authorization (الإذن)

⊙ Give Permissions to User
⊙ Admin and User Panel and their Access

(*) We can make our own authentication tool or use already present tool.

⊙ Identity:

→ Microsoft provided it for authentication and authorization of User.
→ Different roles and policies system in authorization.

→ Implementation and Integration:

⊙ MVC Project → Authorization → Authentication Type → Individual Accounts ← Create

⊙ Areas Folder → Identity → Pages (Default Identity Razor Pages)

⊙ Dependencies → Packages → Identity Pages class

(*) (1) Identity is implemented as class Library and (2) Razor View.

⊙ Register and LogIn already created.
(Add EF- Packages)

⊙ App settings.json (DB connection string)
(Put string here, so it will not change in code) (change DB name already created before)

→ Apply Migrations → Register confirmation
(Tables Default created)

→ User Tables (check Id, Name etc...)

→ LogIn (check)

→ Add → New Scaffolded Item → Identity (left)
(3rd) (Project)

Install ← Add ← Identity ↵
↳ checkBox → LogIn, Register

→ Views → shared → Layout

DB Context File Select → Add

→ Register.cshtml → Register.cs
(Code File)

→ Identity Main Classes:

(i) ~~Sign~~ Sign-In Manager

(ii) User Manager.

(Register.cshtml.cs)

(We can directly call functions from them).

⊙ Customize Identity:

→ Identity → Pages → Register ↓

Input city ← placeholder=city ← copy ↵

→ cshtml:

public string City {get; set;}

→ Models → Class → MyApplicationUser

public class MAU: IdentityUser

{ public string City {get; set;}

→ DBContext:

... ApplicationDbContext: IdentityDbContext<MyApplicationUser>

⊙ run Migration

:add city to user.

→ Program.cs

→ Identity

Add.Default Identity < MyApplicationUser>

→ Register.cs

Change IdentityUser with MyApplicationUser (ctrl+H (Replace all)

:Don't replace in MyApplicationUser the IdentityUser.

: Error : Using WebApplication2.Models.

-) Register.cs

: On Posting Asyn(

```
if(....)
{ var User = CreateUser();
  User.City = Input.City();
  :
}
```

31/12/24

→ Data Source:

→ File → Database

→ API → Input etc.....

→ Classes → List → Array

- Data source changes, retrieval changes
- LINQ (Language Integrated Query)
  (It is part of language as SQL not part of language).

→ Multiple data source retrieval same syntax.

→ LINQ is converted into other data sources.

Anonymous Function
→ Statement
→ Lamda

• Delegates as pointers in C++

→ In LINQ, there are two types of syntax:

(i) Methods (Extension methods) (ii) Query

⊙ Implementation:

→ Console Application →

```
Main( string [] args)
{
    String[] data = { "Pakistan",
                      "India", "Spain",
                      "USA" };
    Var query = data.where( s =>
                      s.StartsWith("P");
    foreach (var item in query)
    {
        Console.WriteLine (item);
    }
```

: using string with query

→ Product class.

```
{
    Name
    Description
    Category
}
```

```
                                        new List<Products>
List<Product> data = { new Product {
                        Name = "P1", Descrip = "01"
                        Category = "C" };
                        (make 4)

Var query = data.where(s => S.Name.Starts
                            With("P"));

for each (var item in query)
  { Console.WriteLine($" {item.Name},
                    {item.Descrip}, {item.
                                    category}");

-> var query = data.where(s => S.Name starts
                                With("P"))
            .select(p => p.Name

-> class Temp
    {
    }       string Temp1{ }
              Name
            string category [ ]

: var query = ---------
            .select(p => new Temp {
                    Name = p.Name, Category
                                = p.Category});

: We make class again and
  again for selection
  so we use anonymous object.
```

→ Anonymous Class

```
var query = ----------
        .Select(p => new { Name =
                    p.Name,
                Category = p.Category });
```

: object type : a (Anonymous)

```
query = query.Where(x => x.Name.
                EndsWith("N")).
                ToList();
```

(Allow execution)

→ Navigational Properties:

- One to many.
- Many to one

→ string sql = "Select * from products JOIN ..."

(Lengthy way)

: var data = dbContext.Category.
Where(x => x.Name.Equals
("Electronics")).
Products.ToList();

(without using JOIN).

: var data2 = dbContext.Products.First
(s => s.Id == 2).Category();

⊙ Identity in MVC:

→ Views instead of Razor Pages

→ Controller (- userManager,
- signInManager)

→ buider.Serivices.Add Default Identity...

→ Synchronous Programming
(Task Execution in sequence)

→ Async programming
(Allowing other program to run independently of execution of other program)

⊙ Task (Data will be returned but in future)

2/1/25

⊙ Authorization:

→ Give access to Users according to role. (Admin, User).

→ Two Methods.

(i) Role-based
(Assign every User a role and define its access).

(ii) Claim-based
(Users can have some claims)

⊙ Authorized tag (Only accessible to authorized user else to everyone)

→ [Authorize (Rde = "Admin")]

o Policy and claim based Authorization

→ Claim based.

↳ Basically Key-value pair
like I am SE student,

↳ Its general, claims are according to the scope of Application

↳ Both roles and claim will be defined.

↳ Rde basically also becomes claim

↳ When we register a user, we also assign's some claims.

→ Claim base or policy base used for authorization.

⊙ Implementation:

bilder.Services.AddDefaultIdentity...

→ Program.cs

builder.Services.AddAuthorization
(options =>
{

: Claim already in Identity.

options. AddPolicy("BusinessHoursOnly",
policy => policy.RequireAssertion
(context => DateTime.Now.Hour >= 9
&& DateTime.Now.Hour < 11));

⋮

→ Controller.

: [Authorize(Policy="BusinessHoursOnly")];
public IActionResult Privacy()
{
return View;
}

→ Register.cshtml.cs:

var claims = new List<Claim>
{ new Claim(ClaimTypes.Email, Input.Email),
new Claim("department", "CS")
};

await _UserManager.AddClaimsAsync(
User, claims);

Var UserId = await _UserManager.Get
UserIdAsync(User);

→ options.AddPolicy("DepartmentAccess",
policy ⇒ policy.Require
claim("department"
(Key)
⊙ [Authorize(Policy = "claim("department",
"Departmant")] "Cs")

→ Valid Email Policy:

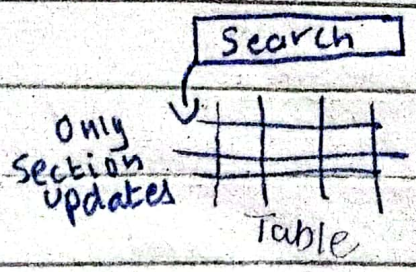
: options.AddPolicy("EmailValid",
policy ⇒ policy.RequireClaim
(ClaimTypes.Email))

⊙ Admin LTE (Admin Panel Template).

⊙→ Complete Page Reload on every request till now

⊙ **Partial Upload:** (updates)

AJAX
(can be used to get and post data)



→ we can send data from server to client in Json form.
(Data format)

→ JQuery:

Library of Functions

```
<div>
    <input type='text' id='txtData' />
    <input type="button" id="bl" value="
</div>                                  Load value
<div id="partialPlaceHolder" style=
www root \ lib                          display:none">
```

→ `$(document).ready(function(){`
(JQuery code start)

```
    $("#bl").click(function(){
        var data = $("#txtData").val();
        $.get('/Home/MyAction', { InputData:
.Data, Url                              Data },
                                        (Json Format)
            function(result){
                $('#partialPlaceholder').html
                    (result).fadeIn
                    ('slow');
    });
```

→ Controller

```
public ActionResult MyAction(string x
{   var s = "This is some data";          result)
    return PartialView("_myPartialView", x);
```

→ Partial View:

 This is my partial view

 Partial View

→ Syllabus:

→ MVC → Partial Views

→ Docker (Not included) → View Component

→ TempData, ViewData, ViewModel

→ DI → Identity (Authen, Author)

→ Repositories