WEB ENGINEERING

BSE: 5th Semester

19/9/24

- .NET Language and C#
- After mid project (Individual)
- Basic C#, Web (5-6) Lectures, HTML and CSS (Bootstrap) and Javascript.
- Front-end, Backend (ORM) etc....
- **Web:**

  Client - Server architecture
  (Browser) (Receive request)
  (Generate request) (After processing performs operations)

- Static and Dynamic
  (Not changes) (Content changes) (Generate HTML on runtime)
  (Wikipedia) (HTML) (Facebook) (Any Backend language used)
- Parts of website:
  (i) Admin (ii) User
- Response always return in HTML form.
- Behaviour in website is done by Javascript. CSS, HTML and Javascript will be returned for dynamic behavior.

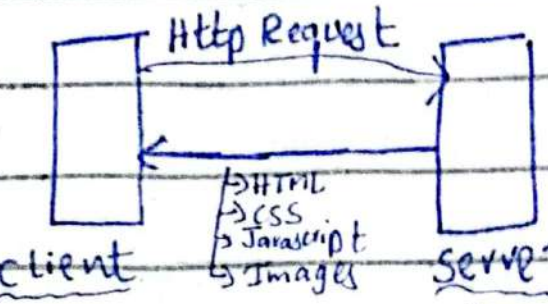
(*) HTML (content), CSS (Style) and Jarascript (Behaviour)

(*) To uniquely identify the system, we need its IP address.

(*) DNS (Domain Name Server) contains IP address. We can directly get to website by IP address.

⊙ Web Architecture:

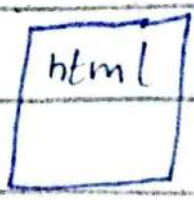
(*) (Generates Http address)



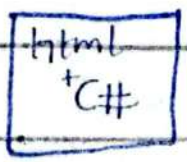
: Email (SMTP)
: File (FTP)

client → web browser
→ Program / Code
→ Tool

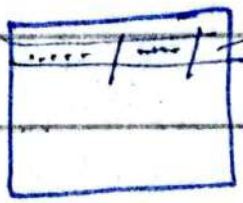
server → Web Server : 80 (Door to enter) ← port number

(*) 

Static Page
(My page.html)



Dynamic Page
(JSP, PHP, asp)

(*) 

Routing Page
(other Pages)

* Current Date and Time:

<h1> Current time : @ Sytem.DateTime.Now </h1>

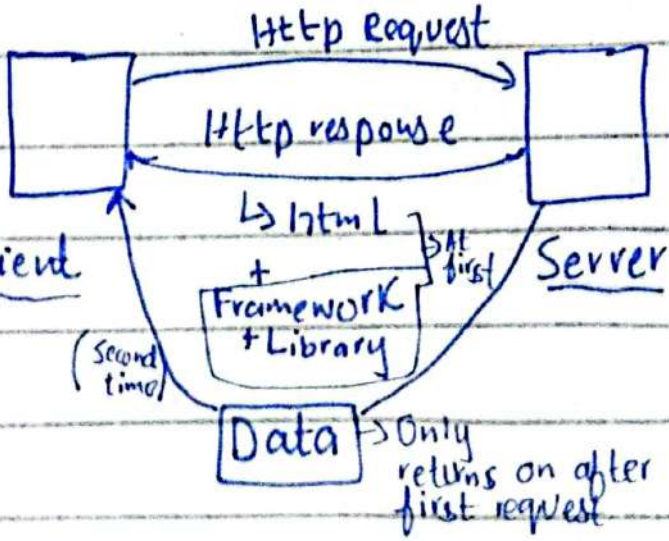
(This page only returns HTML)

→ Client only reads HTML:

<h1> Current time: 11 :16 </h1>

(Browser only reads HTML)

* DB → Data → HTML → Browser.
* Single Page Application (Complete Application has only one page) (Facebook etc....) (HTML generation is at client end). (DB is at server end)



* C# (Not only for web) (Also for mobile app development etc.....) (Broader set of technologies are covered in it).

C# (OOP) (classes) (objects).

24/9/24

* JSON (Format of Data) (Key value form) (From client to server).
* 56 lectures on C#.
* .NET Framework (works on windows early). .NET Core (Can work on Mac, Linux etc....). (For cross platform). (Extension .dll for cross platform) (Extension .NET exe. file for windows)
* .NET
  - FCL/BCL (Framework Class Library) (Different set of classes for different purpose form) (Console class)
  - CLR (Common Language Runtime)
  - C# (Csharp)
* Native code (Machine dependant) (Intel code not works for AMD)
* Intermediate code → Native code
  Common Language Runtime (CLR)

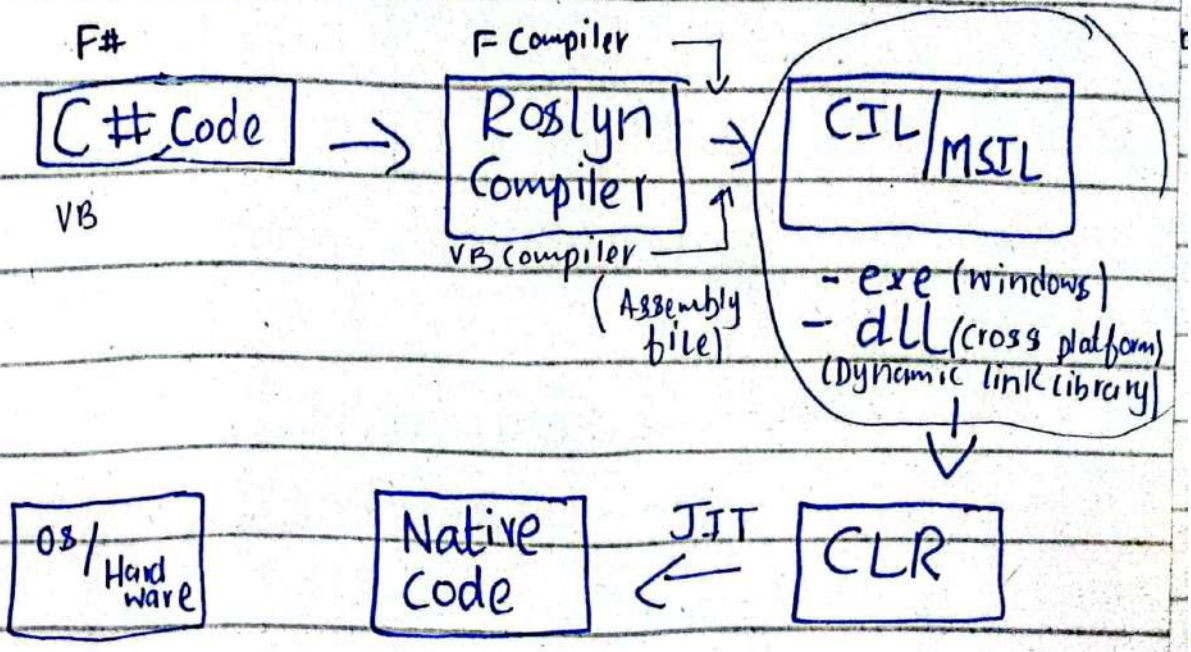
CIL: Common Intermediate Language
MSIL: Microsoft

* CLR uses JIT (Just In Time) Compilation to run/convert part of code not complete.

Roadmap

* Cs File → compile → File (Exe, DLL)
Native code ← JIT ← CLR

*



⊙ Classes:

internal (Default Access Modifier)

```
Class Weather {

}
```

* Make functions as library to be used in many projects. (calculator).

* Internal (Only visible inside the project). We will make it public to access others.

* Internal (Accessible to one assembly only)

Public (Accessible to other assemblies).

* class Weather {

```
    int weather;
    string city;
    public void print()
    {
        Console.WriteLine("City: "+
            weather);
    }
}
```

* Main () {

```
    Weather w = new ();
    w.print();
```

: Main Method is already built in new Compilers

* Concatenation:

```
Console.WriteLine($"The {City}
    weather is {weather}");
```

* Read only : Getter
  Write only : Setter

Both: Getter, setter.

(*) Property of variable/attribute:

```
public string City {
                Property First letter capital
    get { return city; }   City
    set { city = value; }  City = "Lahor
}

public int Weather {
    get { return weather; }
    set { weather = value; }
}
```

(*) Namespace:

Box (Multiple Files inside)

→ To differentiate set of classes on basis of functuality.

(*) Object Base class from where every class is inherited.

```
:internal class weather : System.object
                          (By default added)
    {
    }
```

⊕ Tostring override: Explicitly.

```
public override string ToString()
{
    return $"City : {city} , Weather:
                {weather} ,
}
```

same code run

```
console.writeLine(Weather);
console.writeLine(weather.Tostring);
```

26/04/24

* STREAM:

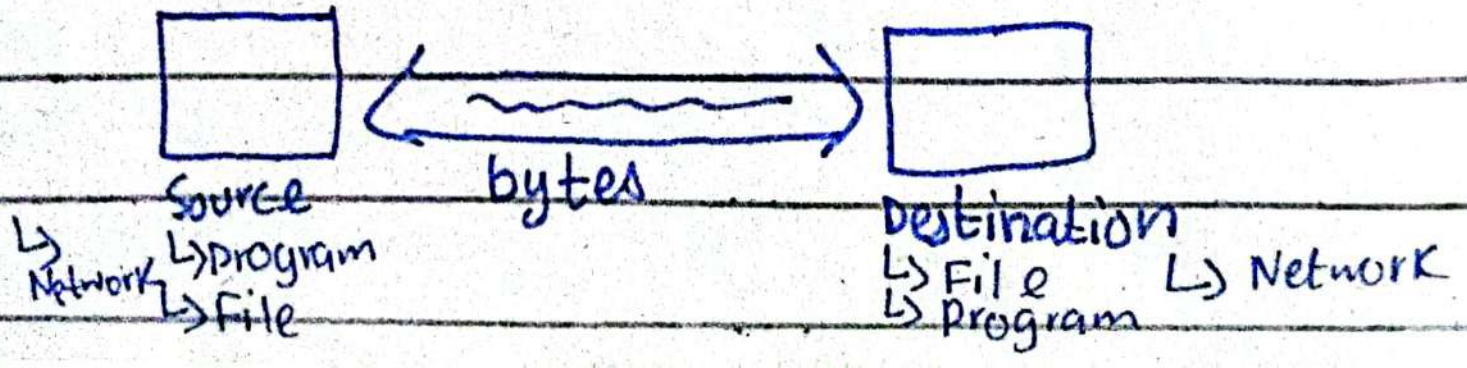
Transfers data from one place to other. (Source to Destination)

* It is transferred in form of bytes.

Suppose Source (Program) → Destination (File)

(can be reversed)

* Flow of bytes from source to destination (pipeline)(stream).



:String:
Byte oriented Character oriental

(*) Include namespace of stream:
:- Using system.io (Library/Namespace Included)
(System.IO)

(*) static void Main(string[] args)
{
FileStream fin = new FileStream ("file.txt", FileMode.create)
fin.WriteByte((byte) 'A');
fin.Close();
}

(*) for (int i='A'; i<='Z'; i++)
{
fin.WriteByte((byte) i);
}
fin.Close(); : From A to Z

Byte oriented:

(*) Open Mode:
{
FileStream fin = new FileStream ("file.txt", FileMode.open);
int data = fin.ReadByte();
Console.WriteLine(data);
file.Close();
}

→ Complete File Read:

```
int data = fin.Readbyte();
while (data != -1)
{
    Console.WriteLine(data)
    data = fin.Readbyte();
}
fin.Close();
```

⊛ Copy File:

```
FileStream finCopy = new Filestream
                    ("filecopy.txt",
                     FileMode.create)
Filestream fin = new Filestream
                    ("file.txt", FileMode.ope
int data = fin.Readbyte();
while (data != -1)
{
    finCopy.WriteByte((byte)data)
    data = fin.Readbyte()
}
fincopy.Close();
fin.Close();
```

⊙ Character Oriented:

(High level data can be saved without converting into bytes)

: Automatically converted into bytes

```
static void Main(string[] args)
{   // Character Oriented streams

    string data = "This is some data";
    decimal anotherData = 34.564 ;
                                (34.564)

    FileStream fin = new FileStream
                    ("file.txt", FileMode.
                                create)
    StreamWriter sW = new StreamWriter
                                (fin)
     sW     .WriteLine(data);
     sW     .Close();
    fin.Close();

=> StreamReader sR = new StreamReader
                                (fin);
    string Line = sR.ReadLine();
```

(*) Product Class :

```
internal class Product
{   // Auto Implemented Properties
    private int id;
```

public int Id { get { return id; }
set { id = value; }

* variable automatically made by compiler

public string Name { get; set; }
public string Description { get; set; }

(*) internal class ProductServices
{
public void Add(Product product)
{ // add this product into file
/*
id, name, description
1, laptop, this is a laptop
2, mobile, this is a mobile
*/
Console.WriteLine("Enter id: ")
int id = ReadLine();
Console.WriteLine("Enter name: ")
string name = Console.ReadLine();
string descrip = Console.ReadLine();
FileStream prod = new FileStream
("product.txt",
FileMode.append

get= split {id, name, descrip}

```
StreamWriter SW = new StreamWriter
                        (bin);
sw.write( $"{id}, {name}, {descrip})
bin.close();
}
```

④ Add(Product product)

```
{
    string data = $"{product.Id},
                  {product.name},
                  {product.description}
StreamWriter sW =new streamWriter
                  {"data.txt", append}
StreamWriter.WriteLine(data);
streamwriter.close();
}
```

⑤ public Product get {int id}

```
{   string data = "1, mobile, this
        is a mobile description
string[] productData = data.split
                        (",");
}
```

```
public List<Products> GetProducts
{
    Do it on compiler.
}
```

1/10/24

(*) Write and Read objects.
(Serialization) (JSON format).

(*) Internal class Product

```
{
    static void Main(strings[] args)
    {
        Product product = new Product();
        product.Id = 1;
        product.Name = "Test";
        product.Description = "ABCD";
        StreamWriter sw = new StreamWriter
                          ("data.txt", append: true)
        string jsonProduct = JsonSerializer.
                             serialize(product);
        sw.WriteLine(jsonProduct);
        sw.Close();
    }
}
```

: using System.Text.Json

{ Key : Value } : Format in File

(*) Reading:

:Deserialize
```
StreamReader sr = new StreamReader
                        ("file.txt")
string data = reader.ReadLine();
product product = JsonSerializer.
                  Deserialize<Products>
                        (data);
Console.WriteLine (product.Id);
Console.WriteLine (product.name);
Console.WriteLine (product.description);
```

(*) In Product class:

```
public category category { get; set; }
                (id, name)
```

→ product data enter
→ category data enter
→ Then write it to file.

(*) :New category class.
```
Product product = new Product();
product.Id = 1;
product.Name = "Computer";
product.Description = "ABCD";
category.category = new category();
```

```
category.Id = 1;
category.Name = "Electronics";
product.category = category;
StreamWriter sw = new StreamWriter
                  ("data.txt"; append:tru
string jproduct = Jsonserializer.
                  SerialLize (product);
sw.WriteLine (jproduct);
sw.close();
```

(*) Control the property to write or not:

[In Product class]

```
[JsonIgnore]
: Id property
```

: Not be included in writing.

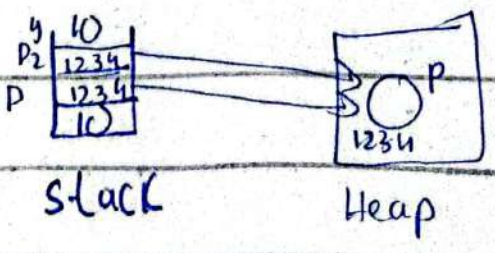
(*) Data Type:

(i) Value Type (Memory on stack)

```
; int i = 10
```

(ii) Reference Type (Memory on heap)

```
Product p = new Produ
Product p2 = p
int y = 10;
```



→ class Product
{
public Product (int id)
(constructor) (we can make for others also)
}

→ Object Initializer syntax:
(curly braces) (use default constructor and set properties)

Product product = new Product {
Id=1, Descrip="ABCD"
};

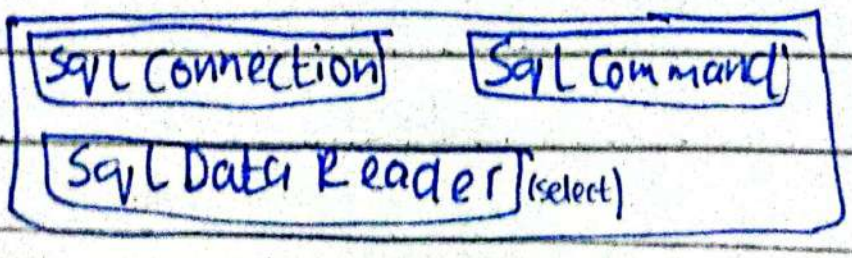
: public Product ( )
{
}

: Product { Name= "Mobile" };
: Only default constructor will work for every property.

(*) product product = new(); (This can also make object).

(*) Database Use:

→ ADO.Net is used to access Database table.



(*) SQL Server by Microsoft.
In visual studio:
View (SQL server Explorer).
Visual studio Installer
(Data storage and processing)

(*) Database Right Click →
Add new Database → DB Name
. New Table → Right Click Table
New Table

(*) Create Table
{
[Id] INT NOT NULL PK,
[Name] varchar(50),
[Description] varchar(50),
};
→ Click Update Table → Update Datab
→ Add New data.

(*) Product product = new Product
{ Id=1, Name="Mobile",
Description="ABCD";

: Solution Explorer
Double file name
(check for Package)

→ For using SQL;

```
using Microsoft.Data.
      SqlClient;
```

Microsoft.data.sql (Client).

→ After Product initialization:

```
SqlConnection conn = new SqlConnection
                     (connectionString);
string connectionstring = " ";
```

(Right click on database)(Select Properties)
(connectionstring property)

: Till Link (True) Integrated Security (No need of username or password)

```
string query = "select * from products";
```

(Add data before in table).

```
SqlCommand sqlCommand =
        new SqlCommand(query, conn);
SqlDataReader reader = sqlCommand.
                       ExecuteReader();
while (reader.Read())
{
    Console.WriteLine(reader.GetString(0),
                      " ", " ", .GetString(1));
                      " ", " ", .GetString(2));
}
conn.Close();
```

3/10/24

(+) Microsoft Documentation (Reading) (Book Reading)

(+) Serialization (Object Conversion into other forms) . (JSON, XML, Binary)

(+) User select any category (1,2,3) . Data in Category name, Product name, product Id, product description.

-> Console APP:

Category

Id (PK)	Name
1	"
2	"
3	"

Product

Id(FK)	Name	Description

⊕ string connectionString = " ";
SqlConnection conn = new SqlConnection
(connectionString)
string query = "Select category.Id', product.Id, category.Name,
product.Name, product.description
from category join product
on category.id = product.id
where category.name in
(name1, name2, ...)"

SqlCommand sqlCommand =
new sqlCommand(query, conn);

SqlDataReader reader = sqlCommand.
Execule Reader();

while (reader.Read())
{
Cousole.Write(reader.Getstring(0));
Cousole.write(reader.Getstring(1));
" " " .GetString(2));
" " " .Getstring(3));
" " " .Getstring(4));
}
conn.Close();

④ **Insert:**

> For Using Network Server for SQL. (Add Network).

```
{
// Connection establish;
// Getting Data from user
Console.Write("Enter id;");
string Id = Console.ReadLine();
// Do it for name1, description.

string query = $"Insert into
        products (id, name, description)
        values ({id}, {names}, {descriptions}"
sqlCommand = new sqlCommand
        (query, conn).
conn.open();
int count = sqlCommand.
        ExecuteNonQuery();
Console.Write($"Number of rows
        inserted: {count}");
conn.close();
}
```

→ We set id as auto-increment
(Right-click table → View Designing
→ Identity Specification ← Column Properties ← Id
(Make True) (Identity Seed : start value)
: Now id will not change by us.
: Now we will not take id by us as input.

④ Update:

```
Console.Write("Enter id: ");
string id = Console.ReadLine();
string name = Console.ReadLine();

string query = $"update product
                set name={name} where
                id = {Id}";
// Count method
SqlCommand sqlCommand = new
                SqlCommand(query, conn);
conn.Open();
int count = sqlCommand.Execute
                NonQuery();
Console.WriteLine($"Number of rows affected
                : {count}");
conn.Close();
```

⊙ Delete :

Same as update from table.

```
=> Console.Write("Enter id you want
                  to delete : ");
string id = Console.ReadLine();

string query = $" Delete from product
                  where product.id = {id}

// Count Method
```

⊙ Exception Handling :

```
Main()
{  try
   {
     // Any code
   }
   catch(Exception ex)
   {
     Console.WriteLine("Some error
                        occured");
   }
   finally {
     conn.Close();
   }
```

: After exception finally work

(*) Proper code:

→ Readable → Proper / Meaningful Name

→ Comments added.

⊙ Application Debug:

→ Debug Point (It will break at that point)

(F10 for next line)

(In Local Window variable name, value etc.

(Manually add value in Vase)

to be observed

⊙ Aggregate Value:

→ single value (select statement)

(Min, Max, Avg)

→ Function:

```
Execute Scalar();                    : object type
}
query = "selat count(*) from products;
SqlCommand ------
conn.Open();
object count = sqlcommand.ExecuteScalar();   (return objects)
Cousde.WriteLine($"Count: {count}");
}
```

8/10/24

(+) Cousole → (Interface) (Front-end changes) Web Application

⊙ Class Library:

To use at multiple places some functionalities.

→ Solution → Rigut Click → Add New Project → Class Library (C#)

→ DAL (Manages functionality).
(Separate file will be made)
(Assembly Class) (Only visible in their assembly).

→ Dependency → Rigut Click → Add Project Reference → My DAL

(*) CS project have all the dependencies (By double clicking on project).

(*) User authentication function.

```
void UserAuthentication(int uName, string pass)
{
    // connection
    string query = $" select * from users where userName = '{uName}'
        AND password = '{pass}';

    SqlCommand cmd = new SqlCommand(query, conn);

    conn.open();

    object count = cmd.ExecuteScalar();

    if(count == 0)
    {
        Console.WriteLine("Error! Username or password not correct");
    }
    else
    {
        Console.WriteLine("Verified correct Username and password");
    }
    conn.close();
}
```

We can also use DataReader option

```
if(reader.HasRows)
{ "authenticate" }
else
{ "No authenticate" }
```

Single can be handled by taking separate query

⊛ SQL Injection:

⊛ Enter Name : abc
Enter Password: qwerty' OR 1=1 --
(Authenticating) (query' OR 3=3 --)

: By making it concatenation Injection is possible.

: Now we will use parameterized queries.

: select * from users where username = @U and password=@p

: SqlParameter sqP1 = new SqlParameter ("U", username)

// // sqP2 = // ("p", password)

// Add into SqlCommand.

→ Now NO query will be injected.

Command. Parameters. Add with Value ("U", username)

Command.Parameters.Add(sqP2);

:(Both Add and Add with Value do the same work)

: 1 Day More (Class Library and Parameterized) in Assignment

④ Insert Parameterized

```
{
    // Connection establish
    // Getting user and password

    string query =
        "Insert into Users (user, password)
        values (@u, @p)

    SqlParameter sqP1 = new SqlParameter
                        ("u", username)
    //    //    sqP2 =    //    //
                        // ("p", password);

    SqlCommand cmd = new SqlCommand
                        (conn, query);
    cmd.Parameters.AddWithValue
                        ("u", username);
    cmd.Parameters.Add(sqP2);

    SqlDataReader .....
    Writer

}
```

(*) Variable Type:

→ int x=0; : No Type changes : Compile Time

// Implicit type variables;

var y = 0; (int)

var x = "a value"; (string)

var a; X (Not possible)
(Because data type not specified)

y=" "; X
(It will remain same in whole program as type in intialization).

→ dynamic a=null; : Type changes : Run Time

(It can be return type)

a = 1;

a = "Some string";

: Loop

→ foreach(Type varname in List)
{
}

E.g: foreach(string[]s in v)
{
}

→ public int getMin(int x, int y);
" " " (int x, int y, int z);
" " " (int x, int y, int z, int q);
(Multiple overloading)

Solution:

→ public int getMin(params int[] data);
:(More than one params cannot be used) (But multiple variables can be used with params at last)

→ public int getMin(string x, string y, params int[] data)
(It will be at last)

: Default value can also be passed to variables

→ public int getMin(string x, string y="ab", float f[] data);

* In memory table creation and table Management.

Disconnected Model:

=> DataAdapter
Data set
Data Table
(Memory representation of data from DB)
} -> More classes in ADO.NET

* Now Data Table receives requests in memory and provide memory representation of data from DB. (In-Memory Table)
* Limited Data will be carried to memory to manage
* Data Set will be used to receive data table. Now it will be In-memory database.
* DataAdapter will sync In-memory Table and In-memory database and made changes in it. (It will reflects the changes of Data Table in Original Database & System).

(*) First focus is on In-memory table. (Not In-Memory Database yet).

(*) Console.App and then use namespace:

-> using System.Data;

-> Main()

{

DataTable dt = new DataTable();

DataColumn Id = new DataColumn ("Id", typeof(int))

// Name = // // ("Name", typeof(string));

Auto-Increment PK

Id	Name
1	-
2	-

dt.Columns.Add(Id)

dt.Columns.Add(Name);

Id.AutoIncrement = True;

Id.AutoIncrementSeed = 1;

Id.AutoIncrementStep = 1;

dt.PrimaryKey = new DataColumn[] { Id };

Insert: // Get UserName (Console.Write, ReadLine())

DataRow r1 = dt.NewRow();

: Both by index and Name.

r1[1] = "Ali"

: In-memory table.

r1["Name"] = n;

dt.Rows.Add(r1);

→ // Print Data:

foreach(DataRows row in dt.Rows)

{

Console.WriteLine(row["Id"]);

// (row["Name"]);

→ Get Data:

(i) Based on Primary Key:

: Find (Get data on basis of PK)

DataRow dR = dt.Rows.Find(1);

dR["Name"] = "new name";

(Update database new value).

→ Delete Data:

dt.Rows.Remove(dR);

dt.Rows.RemoveAt(0);

: dt.Rows.Clear();

: Take Index of row

: To remove all rows data.

→ Get Data:

(ii) Based on Index:

dR = dt.Rows[2];

: DataAdapter
Select
Insert
Update
Delete

(iii) On basis of condition:

```
DataRow [] data = dt.Select(
    "Name like 'A.%'");
```

⊙ Using Data Adapter:

→ Package Install (Microsoft.Data.SqlClient)

: Main()
: Add

```
{
    string connection = " .... ";
    SqlConnection connection ....
    query = "select * from products";
    SqlCommand command = .....

SqlDataAdapter adapter = new
                    SqlDataAdapter();
adapter.SelectCommand = Command;
DataTable dataTable = new DataTable();
adapter.Fill(dataTable);
foreach (DataRow row in dataTable.Rows
{
    Console.WriteLine($"row[0], row[1]
                    row[2] ");
}
```

: use query of select command property and fills dataTable

: Print Data.

: Debug dt. (DataTable properties)

(*) Insert:

```
DataRow dataRow = dataTable.
                    NewRow();
dataRow[1] = "a new product";
dataRow[2] = "a new product
                description"
dataTable.Rows.Add(dataRow);

string insertQuery = "Insert into
        products(name, description)
        values
SqlCommand insertCommand(insertQuery,
                connection)

SqlDatAdapter adapter = new
                SqlDataAdapter();
adapter.SelectCommand = command.
adapter.InsertCommand = insertCommand;

SqlParameter nameParameter=
        new SqlParameter("n",
parameterName,
SQLDBType,
size, sourceColumn
                SqlDBType.Varchar, 50, "name");
```

Adapter method
: Fill Method
: Update Method
(Insertion, Deletion, Update)

// do also for description.

insert Command.Parameters.Add(name Parameter);

// We wanted to all In-Table values
so we use update:

adapter.Update(dataTable);
// Reflect all changes back to db.

④ update:

(Do it from home)

adapter.UpdateCommand = ??

adapter.DeleteCommand = ??

15/10/24

* Web:

→ Client/Server Architecture

→ Port: 80

→ Basic Language: HTML

⊛ HTML:

⊛ **Heading:** <h>

Paragraph <p>

Division

→ Web Page not only contains HTML.

→ Inline Element (In same line)

→ Block Element (Step by step elements (In different line).

→ (Block level element) (Divide a page)

→ (Inline element) (Container) (On same line)

→ <img>, <h>, <UL>, <OL> (ordered list), <p>

→ To choose our own place of elements we use CSS (Visual appearance) (Position) (cascading style sheet)

→ For behavior in page, we use javascript.

→ Page → HTML
→ CSS
→ JavaScript

→ Form tag which take input from User. It needs to be saved to Server side DB. (Using ASP.NET)

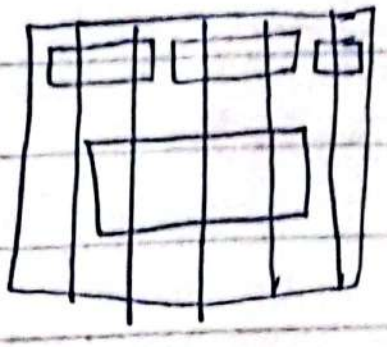
→ Razor Pages, MVC pattern, Single Page Application BLazor (These can be used by page. to make

web application dynamic).

(*) Bootstrap will be used.

(*) FlexBox and Grid style is used for positioning of elements. (Main, Header, Footer). It can also be done using simple CSS.

(*) Bootstrap divides the page into 12 columns.



(*) **MVC** (Model View Controller):

→ One application is divided into 3 parts:

(i) View (HTML) etc.... (Includes Interfaces)

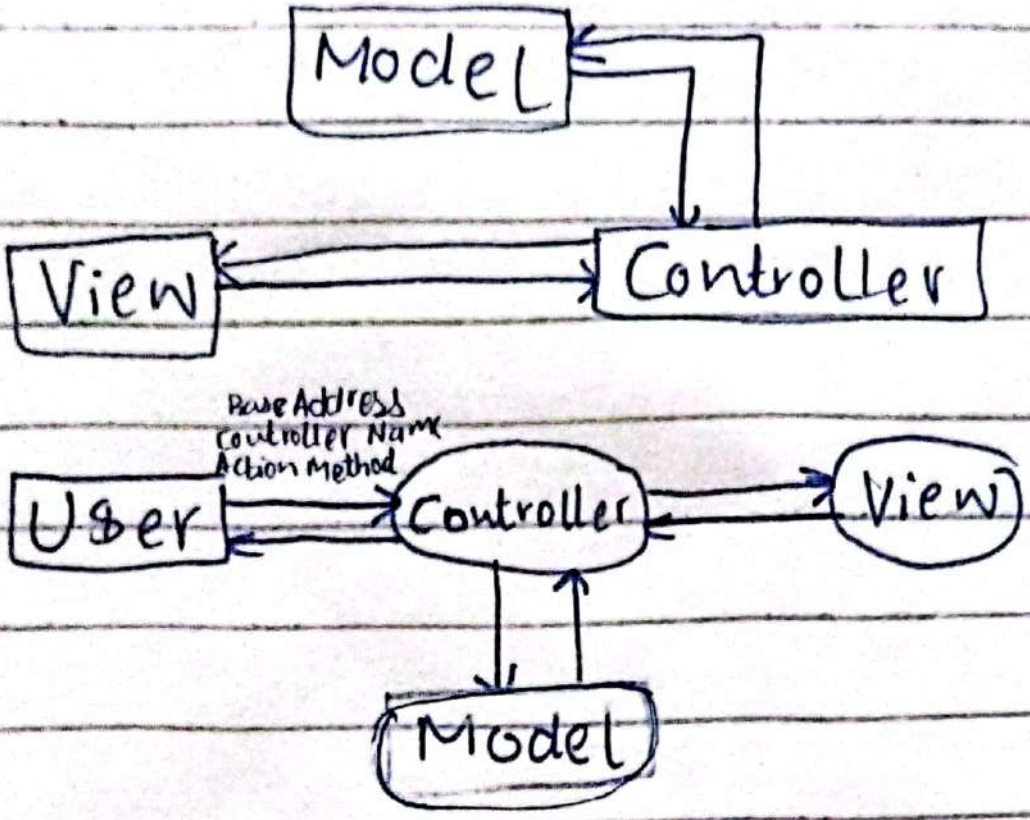
(ii) Controller (Control the flow of application) (Brain)

(iii) Model (Those classes which represent Business) (CMS (Course, Student, Faculty etc....))

; Work is done according to use-case

: Problem contains Entities

(*) MVC Diagram.



(*) Actions in Controller : Add, Update, Delete for student

(*) MVC using : (1) Controller Name (2) Action Method

(*) Multiple controller can be used in web. It is made based on entities. Action Method are defined in Controller.

(*) Search template : (i) ASP NET core App (Razor Pages)

(ii) Model View Controller. (ASP.NET core web App) (Model View Controlle

(iii) Blazor web App

(*) Visual Studio Installer (ASP.NET AND web Development

(Model View Use)

* My First MVC Project

(In solution Explorer) www root folder (css, js, lib)

(site.css) (site.js) (Bootstrap JQuery) (Libraries can be added into it).

* Add -> New Folder -> Images

* Controllers -> HomeController.cs

(Any Name) (This naming convention would be followed) (Student Controller).

* Starts with 'I', its interfaces.

* Class Represents real world object.
(Entities, Functions, Properties)

* Interface:
-> Abstract-level thing which defines what this entity will do.
-> Function Definition.
-> Logger (Application behavior Log)
(For crashing of App) (Real Deployment contains log)

* public string Index()
{
return "My First MVC Application";
}

(Controller) (Action)
-> Home / Index

-> MapController

→ Inspect → Network Tab (Enter)

Response ← ClickOn Action Button

→ public ViewResult About()

New Action method

{

return View();

}

→ Now we will create View

(View Folder in Solution Explorer)

(Home → Add → View → Razor View Empty)

About ←

→ <html>

<body>

<h1> My first MVC Application View </h1>

<body>

</html>

→ BaseAddress / Home / About

(Controller)

→ Model → Add → Class → Product

(Id, Name, Desc, Price) Properties ←

→ New Controller → Add → Controller

ProductController ← MVC Empty ←

: add Project Name:
using MyFirstMVC.Models;
(In Namespace MyFirstMVC)

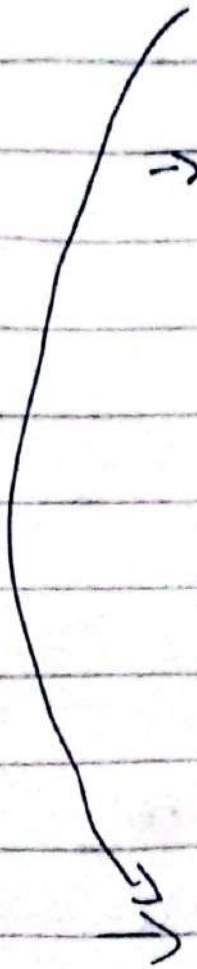
: cshtml (csharp html) (csharp can be embedded)

-> public IAction Result Index()
{
Product product = ...
product.Id = 1;
product.Name = "Mobile"
product.Description = "This is a Mobile"
product.Price = 10;
return View(product); : object modeled
}

=> New view -> Folder (Product)
Index <- Add view
↳ HTML (simple)

(To embed C#) @
<h1> Current Time : @System.DateTime.Now </h1>

: (Browser only receives HTML) (Not Csharp)
: View Page Source (contains HTML only)
: All Processing on Service side.



 @ Model.Name
" " " .Price "
" " " .Description "

: Save changes By red Button.

17/10/24

(x)

return View(product);

Model act as Product

<h1> @ Model.Name </h1>

(f) string mydata: "some data";

return View(mydata); : error will occur because it can receive view not string.

-> Object mydata = "some data";

return View(mydata);

-> We will deal int, string, char as technique:

{
object data = "This is some data";
int id=1;
ViewBag.x = data; ViewBag.y = id
return View();
}

: We get get it as:

@ ViewBag.x

@ ViewBag.y

-> Make Product and category class.

public class Category

{ id
Name
Description
}

```
public IActionResult Index()
{
    Product product = new Product
    {
        Id=1;
        Name= "product1",
        Description="This is a product"
    }

    Category category - new Category
    { Id=1;
      Name="Category 1",
      Description="This is a Category"
    }
```

: we will make new class to send multiple object

: New Folder ViewModels
Category Products ← New Class ↵

: View Model class and Add Product and category as Properties

```
public class CategoryProducts
{   public Product product
        {get;set;}
    public Category cat{get;set
}
CategoryProducts cp= new CategoryProduct
{ Category= cat;
  Product= product};
```

return View(cp);

→ Now we will get it:

: @model "Catego" WebApplication.Models.ViewModels.CategoryProduct

(Telling what is model of view)
(This view is known as Strongly type view)

<h1> @Model.Category.Name </h1>

" " .Product.Name </h1>

→ public class CategoryProducts

{

public List<Product> product {get, set};

pubic category cate {get, set};

}

→ @Models.Category.Name : Display Product by loop

@Models.product[0].Name

@Models.product[0].Description

@Models.product[0].id

→ public class Product

{

" "

: Do it later /// Category category

}

Controller

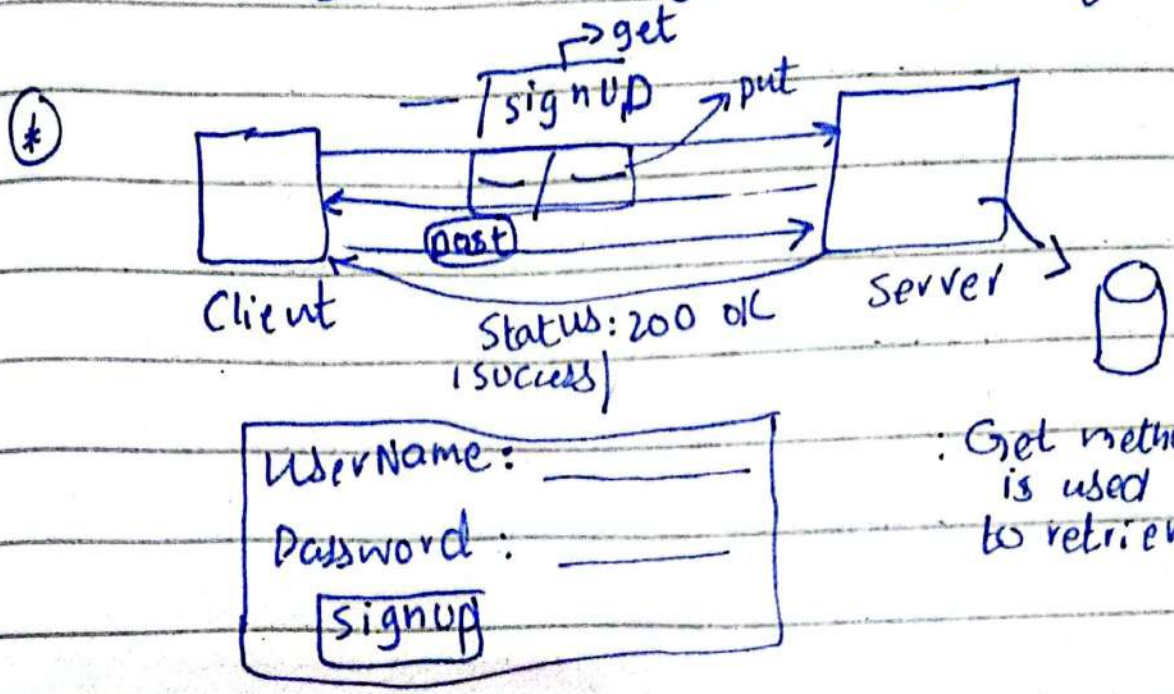
```
List<Product> products = new
                    List<Product> {
  new Product { Id=1, Name="Laptop",
                         Price=123 },
  new Product { Id=2, Name="Car",
                   }          "
     "            "           "
  }
Category Products cp = new Category
                          Products
       { category = category,
  : <div  } Product = product
           return (cp);
       <div>
          <h1> Name: @Model.category.
                            Name</h1>
       <p> @Model.category.Description
                          </p>
       <div>
   <div>

<table> style="border: 2px"
         (optional)
   @foreach(var p in Model.Products)
   {
      <tr>
         <td> @p.Name </td>
          "   @p.Description  "
, table>  <tr> <td>  @p.price     "
```

: Get, Put
Post, Delete

return (product)

: @ model List< WebApplication10.Models.Product>
: @ model List<Product>

<h1> Name : @Model[0].Product info </h1>



: Get method is used to retrieve

① Form Tag :

-> Action -> Method

Action: Controller Name / Action Method
Method: (Post)

-> New User controller :

```
public ViewResult SignUp()
{
    return View();
}
[HttpPost]
public ViewResult SaveUser()
{
    return View();
}
```

: View
New Folder
User
View: signup

: [HttpPut], [HttpDelete], [HttpGet]
(By default)

```
<h1> Sign up </h1>
<forms>
<form method='post' action='User/Save' (SignUp)
    <label> User name: </label>
    <input type="text" />
    <label> Password; </label>
    <input type="text" />
    <input type="submit"
           vale="sign up" />
</forms>
```

→ `<input type="text" name="un" />`

`<input type="text" name="pd" />`

→
```
public ViewResult / IActionResult SaveUser(string un, string pwd) (SignUp)
{
    User u = new User{UserName=un, paLDword=pwd};
    return View()
} (:return RedirectToAction("Index", "Home", new{un}) (AM))
```

:
```
public class User
{
    public string username{get;set}
    public string password{get;set}
```

:ctrl + F5

@model User

<h1>@User.UserName</h1>

<h1>@model.password</h1>

22/10/24

⊙ ViewData :

Same as ViewBag

→ To send data from Controller to View.

(*) DB code must be in model and call it in controller.

(x) ViewData["Title"] = "Home Page";

(*) Defining Header and Footer for all Application: (common code)

⇒ Layout Page (In shared Place)

: shared → _Layout.cshtml.

: Viewimport → ViewStart.

: Privacy Page

: Two Pages Layout in Daraz (Admin / User)

: View Page Source

(+) Request sent then layout will return both (shared and other).

→ Layout Page → RenderBody()
(contents will be merged here)

: Define id in footer

: shared Folder → Add → New item → Razor Layout (Layout)

: <body>

: Layout2

<h1> My second layout</h1>

: By default: ViewStart Layout

: Piracy Page:

```
{
    Layout = "_Layout2";
}
```

: Not wanted to use layout:

```
{Layout = null
}
```

: Multiple Viewstart files in different folders.
(For different Pages Management)

(*) User Class:

<form>

: form method="post" asp-action="LogIn"
asp-controller="User"

```
<label for="Username">User Name:</>
<input type "text" name="Username" />
<label for="Password">Password</label>
<input type="text" name="Password" />
<input type="submit" value="LogIn" />
</form>
```

→ public IActionResult LogIn()
{
return View();
}
[HttpPost]
public IActionResult LogIn(string Username, string Password)

: Include : public IActionResult LogIn(User user)
: Model Binding

→ Model → User Repository
(class)

: public class UserRepository
{
public bool void LogIn SaveUser(User user)
{
}
}

: public IAction Result LogIn(User user)

{

UserRepository UR = new User....

bool IsAuthenticate = UR.LogIn

if(is Aumthenticate)

{

return RedirectToAction("Home" /Action/ "Home" /Controller/);

}

else

{ return View();

}

: Controller layer thin (Only control flow)

: Network window (Request/Response)

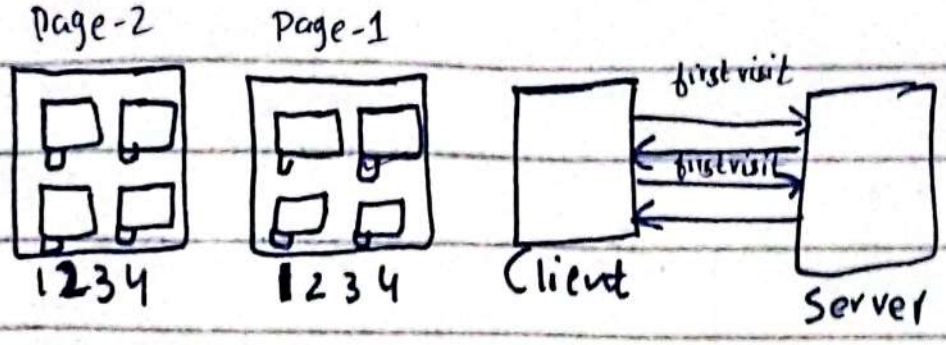
Return RedirectToAction (Two requests return) in response of other

24/10/24

* Stale Management:

→ Web is a client/ server architecture.

⊛ If product is selected on page-1 then we move to page-2 and the selected product on page-1 is unselected. (state not managed)



⊛ Different clients, different requests management required.

⊛ State maintain:

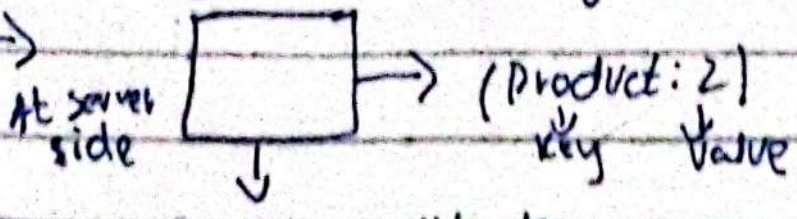
(i) Cookies (At Client Side)

(ii) Session (At Server side)

⊙ **Cookies:**

→ Piece of information

→ Key : value form

→ 

When request is sent, cookies will also be sent and maintained.

In HomeController → Index

HttpResponse
HttpContext.Response.Cookies.
Append("first_visit_datetime", System.DateTime.Now.ToString());

-> public IActionResult Index()

{

if(HttpContext.Request.Cookies.ContainsKey("first_visit_dateTime"))

{

object message = String.Empty;

message = "you visited us at: " + HttpContext.Request.Cookies["first_visit_datetime"];

}

else

{ HTttp Response/ Context..

message = "you visited first time";

}

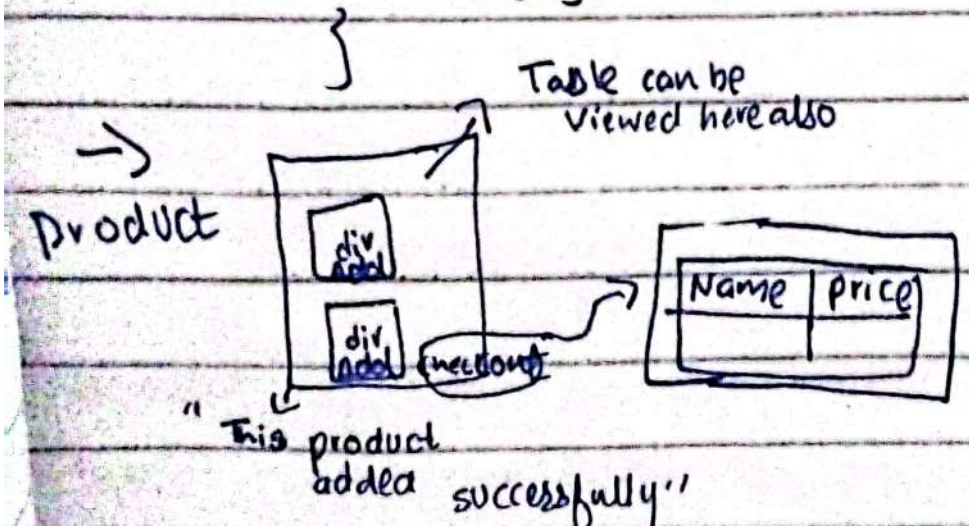
return View(message);

> View -> @Model

=> Page -> Right Click -> Inspect -> Application
LocalHost <- Cookies

=> To keep expiry date of cookies more.

: else
{ CookieOption options= new CookieOption();
options.Secure = true;
options.Expires = DateTime.Now.AddDays(1);
HttpContext (third parameter /options);
message = ----
}



: public IActionResult Index()
{ object message = string.Empty;
if (HttpContext.Request.Cookies.ContainsKey("first_visit_date"))
{ message = HttpContext.Request.Cookies["first_visit_date"];

else

{ HttpContext.Response.Cookies.
append(" ");

};

→ Model class → Product(Id, Name, Price)

Controller → ProductController

: Model Class → Product Repository.

{

public List<Product> p
{ get; set; }

// public List<Product>
GetProducts()

{

List<Product> p = new ...,

p.add(new Product { Id=1,
Name="Mobile",
Price=123})

// //

return (p);

} }

→ Index()

{ ProductRepository pR ....

return View(pR.GetProduct());

}

→ View → Index :

@model List<Product>

<h1> My Ecommerce site</h1>

@foreach(~~var~~ var product in Model)

{

@Product.Name

// - Price

/ "Add to Cart"

→ Product/Index

Add to Cart

:/Product/Index?MyId=@product.Id & @product.name

→ public IActionResult Index(int id)

{

string product = "This product get selected with Id: {Id}";

ViewBag.p = product;

return View();

}

*(Do it) Cookies put and not displayed : JSON format (Serialization)

If(Id>0)

{

//cookies;

}

* ⊕ More Methods of State Maintain (Study by yourself) (Queries, Temp DB)
* ⊕ For content we use HTML and for layout we use CSS.

⊙ CSS:

→ Inline (single Line)
→ Internal (By adding in tag)
→ External (In other file)

(Inline)

⊕ `<h1 style = "color:red"> ..... </h1>`

mypage.html

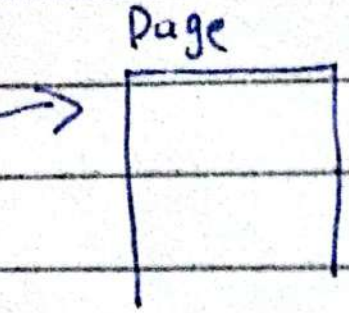
Internal

```
<h1> ..... </h1>
<style>
  h1
  { color:red;
  }
</style>
```

⊕ External

style.css

```
h1 {
  color:red;
}
```



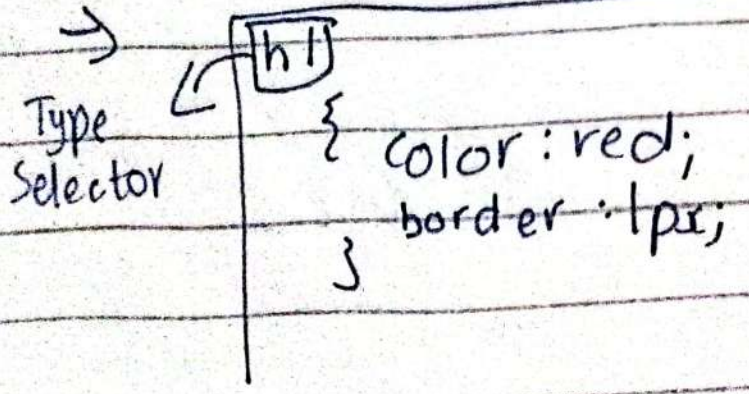
: Id can't be assigned to more than one.

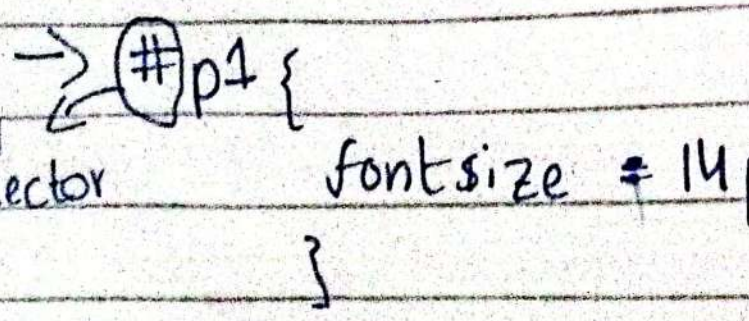
⊙ CSS Selectors:

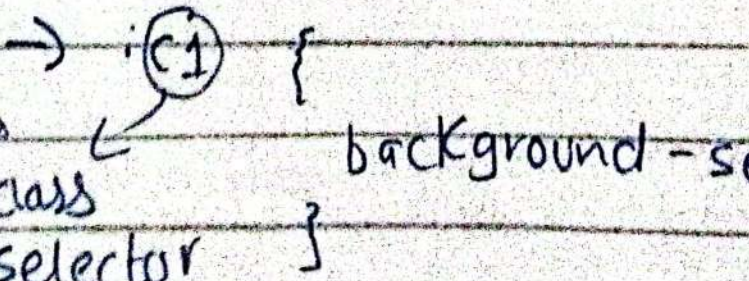
→ Select elements to style or define rules.

→ ———
<h2 class='c1'> ——— </h2>
 .....

→ Attribute, type, id, class (selectors)



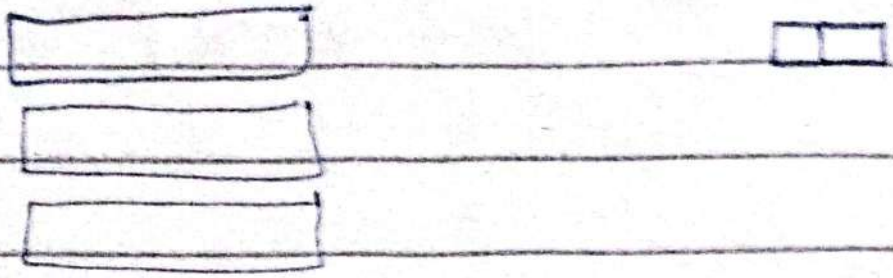




(+) Block elements and Inline Elements

(Next Line / New Line start) (Same line)

( ) ()



(+) For space between box:

→ Box Model

→ Margin, Border, padding

(+) Index.cshtml:

@ { Layout = null }

<html>

Inline > Internal > External

<head> → (2) <style>

</head>

p {
color: brown
}

</style>

<body>

(1) ....

(+) www root → css → site.cs

Add New Item → CSS → Style Sheet

Styles.css

→ styles.css:

```
p {
    font-size: 20 px;
    font-family : "Arial Rounded
                   MT;
}
```

↓ include it.

→ Index.cshtml:

```
<head>
<link rel = "stylosheet" type="text/
      href="../css/style
<p id "p1"> ... </p>
<h1 class="c1"> ... </h1>
<h2 class="c1 > ... </h2>
```

for Id:

→ #p1 {

Background color : yellow

}

→ .c1

{

color : red;

}

→ Inspect → Runtime change (size, color).

⊕ By default box model is initialized

⊕ common style:

→ p, h1, h2 (same style on it)

{

}

→ p.c3

{ color: red

}

.....

→ p#p1

{

}

....

→ #p1.c3

{

}

→ #p1 p (Not works)

type before id/class selector after

→ .c3#p1

{ color: aqua

}

⊕ Index.html

(*) style.css ;

```
a:link {
    color: red;
}
a:hover
{
    color: green;
}
```

(*) Index.cshtml ; ':' (pseudo class)

```
<uls
    <li> This is first element </li>
    //          Second      //
    //          third       //
    //          fourth      //
```

(*) li: last-child

```
{
    color
}
```

(*) p:: first-letter "::" (pseudo elements)

```
{
    font-size :larger
}
```

(complete element)

④ Index.cshtml:

```
<div> This div </div>
```

→ style.css

```
div {
    border: 2px solid;
    margin-bottom: 10px;
    display: Tuline
}
```

(Explicitly making it inline)

31/10/24

① Session:

→ State maintain at server side

→ Middleware Adding

Ⓔ New MVC → Program.CS

```
main()
{
var builder = WebApplication.
                createBuilder(args)
builder.services.AddControllerwithViews();
builder.Services.AddSession();
builder.services.AddDistributedMemoryCache();
```

(1) Builder. AddSession
(2) · Distributed Cache
(3) · use session

→ After app.UseRouting();

→ app.UseSession();

→ HomeController:

→ Index()

:Key: value

{ HttpContext.Session.SetString("mydata", "This is mydata")

→ Privacy()

{ object data = HttpContext.Session.GetString("mydata")

return View(data);

}

→ privacy View:

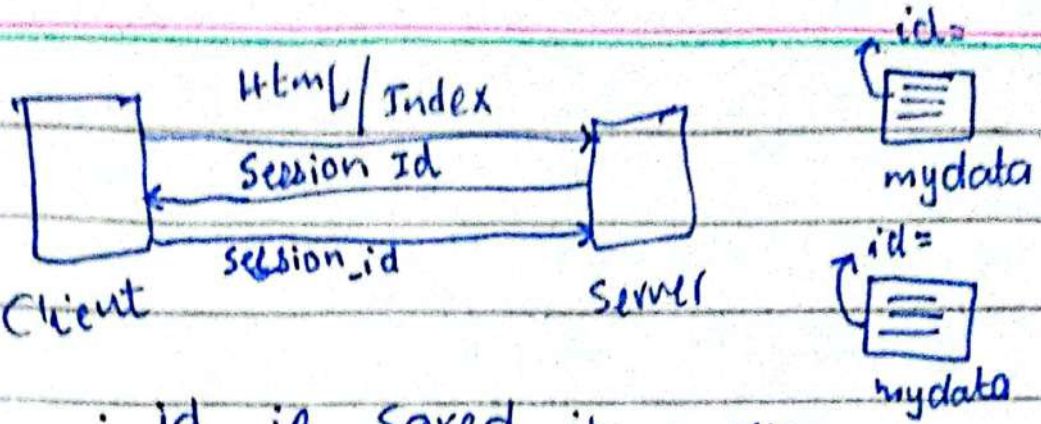
<h1> @Model </h1>

→ Index()

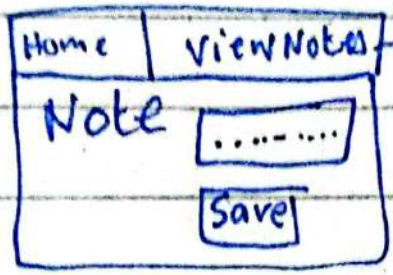
{ HttpContext.Session.Remove(...)
Session.Keys.Contains()
.Session.Get--()
.Session.Set--()

}

: Paper Questions : Inspect → App → Cookies



: id is saved in cookie

(*) 

→ List will be returned

(*) NoteView:

```
<form action="/Home/Note" method="post">
    <label> Note : </label>
    <input type="text", name ="note"/>
    <input type="submit", value="Save"/>
</form>
```

(*) Home controller:

```
AddNote (string note)
{List<String> notes = JsonSerializer.Deserialize
                        <List<strings>>
                        (HttpContext.Session.
                        GetString("data"));
notes.add(note);
HttpContext.Session.SetString(JsonSerialize.Serializer
                        (notes));
```

```
    return View(notes)
}
```

→ Class Note

```
public class Note
{
    public string noteText {get; set;}
}
```

→ NoteController:

```
public ViewResult AddNote()
{
    return View();
}
```

→ View (Note):

View → AddNote

```
<div>
    <form method="post" asp-action="AddNote"
          asp-controller="Note">
        <label asp-for="NoteText">Note</label>
        <input type="text" name="NoteText" />
        <input type="submit" value="save to session" />
```

: using System.Text.Json

→ Note Controller:

[Http post]

public IAction Result AddNote(Note note)

{ if (HttpContext.Session.Keys.Contains("notes")

{ List<Notes> notes = new List<Notes>();

notes.Add(note);

string jsonString = JsonSerializer.

Serialize(notes);

HttpContat.Session.SetString("notes", jsonNotes)

} if(_ _ _ _ _ _)

{

string jsonNotes = Http Context.Session.

GetString("notes");

List<Notes> Notes = JsonSerializer.

Deserialize

<List<Notes>>(JsonNotes);

Notes.add(note);

}

=> List<Note> notes = null

if (_ _ _ _ _ _ _ _)

{ string JsonNotes = ... GetString(notes)

Http ... notes = JsonSerialize.De ... List<Notes>

(JsonNotes)

else { notes = new List<Notes>();

notes.Add(note)

=> View Index:

@model List<Notes>

string jsonNotes =

HttpContext.Sa... JsonSerializer.

Serialize(notes)

SetString

("notes", jsonNotes)

return View();

@foreach (var n in Node)
{
 @n
.NodeText
}

Index (Home)

@{
ViewData["Title"] = "Home Page";
Layout = null;
}

<div class = ... (default index)

Site.css:

```
dir {
    border: 1px solid red
}
```

 This div
 This another dir

```
<head>
<style>
    div {
        boraer: 1 px, solid, red
        height : 200 px
        width: 50%
    } display : Inline-block;
</style>
```

: div has block properties
: span not changes height width
: Inline Block
None

relative, absolute, static

→ display : none (no place or no

display / visibility : hidden

display : Inline

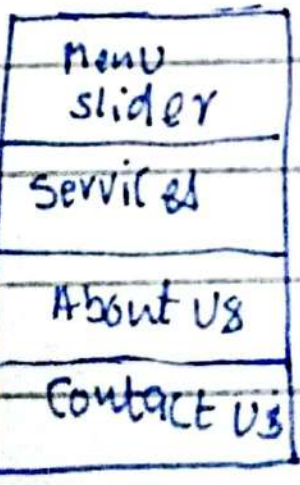
display : Block

5/11/24

* Cookies also present because it differs from session id.
* Inline Display only takes required space. Inline-Block changes height, width etc... but Inline not changes heigh, weight etc...
* Hidden - Take space, None: No space

Partial Views:

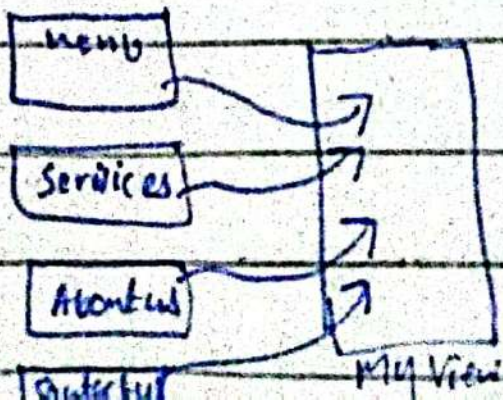
-> we make separate Pages for each so we can make changes easily.



MyView.cshtml

: Fault : Maintainability

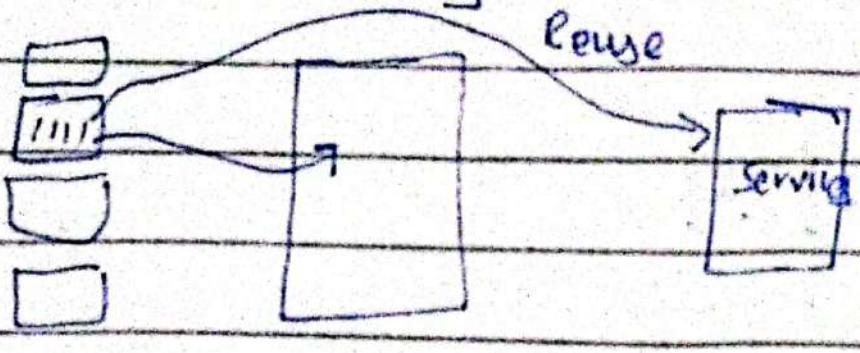
=> separate pages but linked and display together.



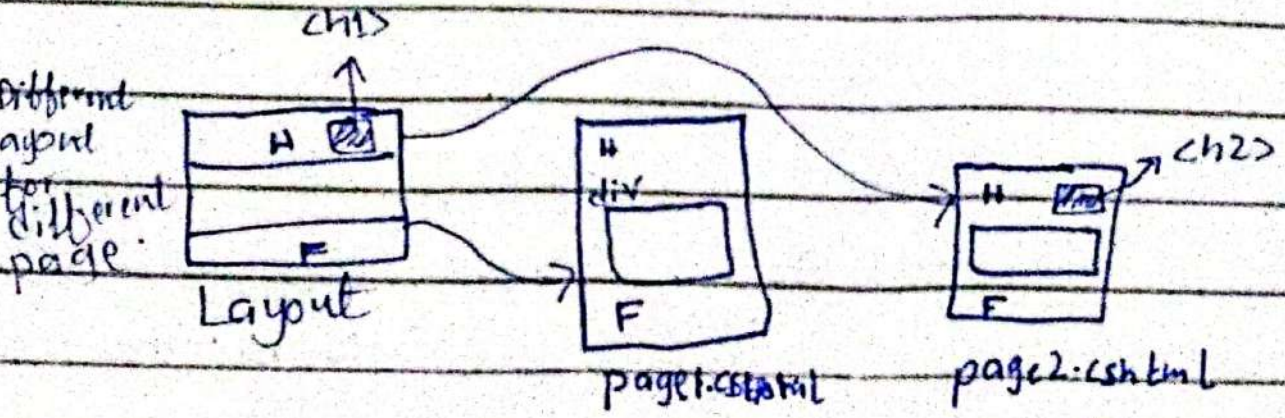
⊛ Benefits :-

(1) Maintain

(2) Reusability.



⊙ Layout Section :



@Section MySection1
{ <h1>...</h1> }

@Section MySection
{ <h2>...</h2> }

HTML Defined in page and merged in Layout

⊛ Default Layout page.

_layout.cshtml

→ @await RenderSectionAsync("Scripts", required:false)

⊛ Home view

→ Index, Privacy

→ Index.cshtml:

```
<div>
    :
</div>

@section Scripts {
    <script src="~/js/Javascript-js"></script>
}
```

:(It will only be shown in Index/.

→ View Page Source
(Our script will be loaded after other scripts).

→ js → javascript file

wwwroot →
js → create file
→ alert(" ")
(Text in js)
→ alert("this is a js file")

* Index.cshtml

*Different html for different page:

```
@section mySection
{
    <h1> Index h1 </h1>
}
```

* Privacy.cshtml

```
@section mySection
{
    <h1> Privacy h1 </h1>
}
```

* _Layout.cshtml
@await ... Render... ("mySection", required: true/false)

(*) Index.cshtml:

```
<div class="text-center">
    <h1> Welcome section</h1>
</div>
    .
    .
    .
    <h1> Service section </h1>
    .
    .
    <h1> Contact section </h1>
```

Now we add partial View:

→ shared Folder → _PR

: We start by underscore

① PR.cshtml

```
<div>
    <h1>Welcome Section </h1>
    .
    .
```

→ index.cshtml:

```
<partial name="_PR" />
```

: Also make it for Service Section and Contact Section.

⇒ Home → AddView → _PR2

```
<div>
    <h1> Service Section </h1>
</div>
```

① Index.cshtml:

```
<partial name="_PR2" model="data" />
```

⇒ Home Controller

```
Index()
{
    List<string> gslist = new List<string>();
    List.Add("
    List.Add("
    :
    return View(List);
}
```

⇒ Index View:

```
var data = "some data"
<partial name="_PR2", model="Model">
```

→ pR2.cshtml:

```
<h1> Welcome section </h1>
@ for each (var services in Model)
{ <span style... > services </span>
}
<p> @ ViewBag.x </p>
```

(+) Practice it and will be task/Quiz in Next class.