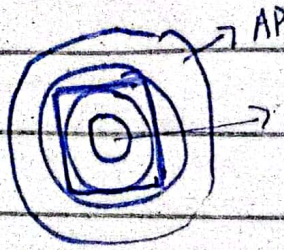
SOFTWARE CONSTRUCTION AND DEVELOPMENT
BSE : 6th SEMESTER

3/2/25

→ Single Page Application
- (Managed at Client-Side)
- (only for data we access server)
- (React, Blazor etc....)

API-Based Programming:

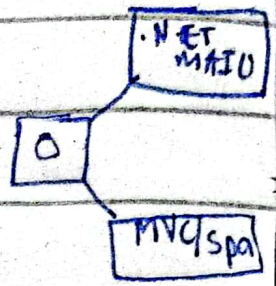
→ Service Based Programming
→ Restful APIs.
→ Realtime (Messaging, notifications etc
→ Socket (source and Destination Channel for real time changes)
(SignalR, Ajax, APIs)
→ Clean Architecture/Onion Architecture
(In form of circles)...



Onion Architecture

→ Ajax, signalR (web API), clean architecture, SPA (Blazor), Dapper {Database} | GraphQL, OData, web security} , Desktop based application — [Window form, WPF, .NET MAUI] , Delegates, Events

→ Microservices (Each service has its own responsibility).



- ⊙ Synchronous
- ⊙ Asynchronous
  - ↳ Kafka
  - ↳ RobbinQ

→ N-Tier Architecture:

→ Different layers used.

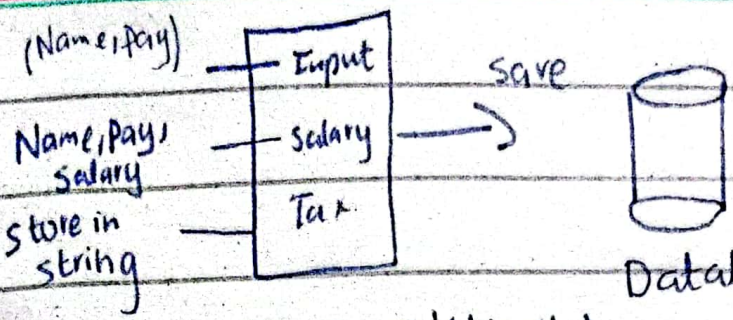
→ Data Access Layer, Business Layer, Presentation Layer.

⊙ N-Tier Architecture:

→ Employee Management System

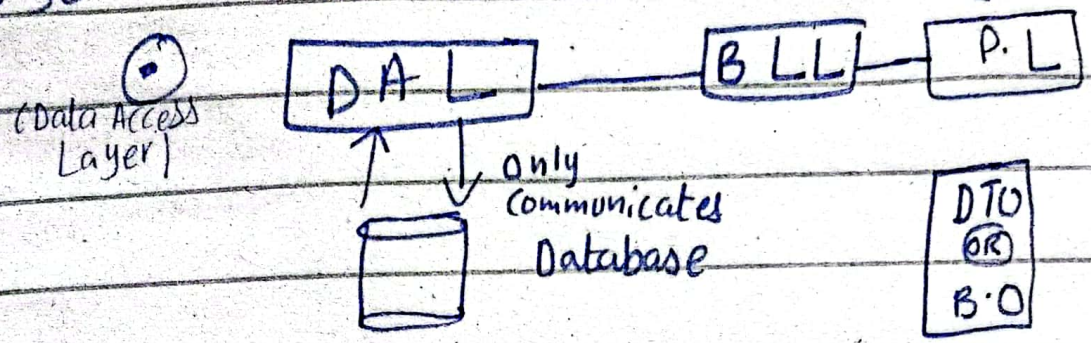
↳ Payroll

0-1000 , 10%.
1000-3000 , 20%.
3000 > , 30%.



(Managing Difficully)

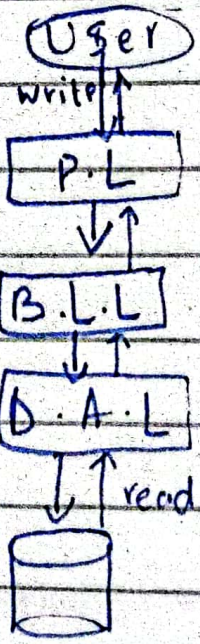
→ So we convert it into layers:



⊙ Different layer interaction uses Data Transfer objects/ Business Objects. (object related properties are present in it).

⊙ Employee (Main Entity)
(EmpNO, Sal, Name, Tax etc.---)

→



: Layers will be followed for writing and reading

: If any business rule is introduced then it will be added in B.L.L function and further no changes.

→ For windows form, we only change the Presentation layer.

→ Against Each Layer we make class Library and then we make exe file.

→ Reference will be from down to top and DTO will be added reference wise in each layer.

⊙ Layers of Employee Data Write:

⊙ D.T.O | B.O:

→ Employee:

- ⊙ EmpNO
- ⊙ Name
- ⊙ Salary
- ⊙ Tax

⊙ D.A.L:

```
class EmployeeDAL
{
    public void Save(EmployeeDTO dto)
    {
```

```
StreamWriter sw = new StreamWriter
                  ("file.txt", true);

sw.WriteLine(dto.name + dto.empNo,
             dto.salary + dto.Tax);

sw.Close();
}
```

⊙ B.L.L:

```
class EmployeeBLL
{
    public void Save(EmployeeDTO dto
                     (Name and Salary only))
    {
        dto.Tax = CalculateTax
                  (dto.Salary);
        Employee DAL dal = new..... ();
        dal.Save(dto.)
    }

    private decimal CalculateTax(Decimal
                                 salary)
    {// statements switches
     :
     return tax;
    }
```

→ class Employee PL

```
{
    public void Input()
    {
        Employee DTO dto = new ... ();
        Console.Write("Enter your name");
        Employee BLL bll = new ... ();
        bll.save(dto);
    }
}
```

⊙ Also do the Read flow and display the Users.

10/02/25

⊙ **SIGNAL R:**

→ To implement real-time in application.

→ Before client is sending request to server. If we update server & server informs about update to client.

→ Socket / Channel for exchanging data between source and destination port.

Source ═══ Datination

Server — Client (Web Browser)

-> SignalR is a library which makes real-time (communication) possible using three ways:

(i) Long polling
(ii) Server Sent Events
(iii) Web Socket

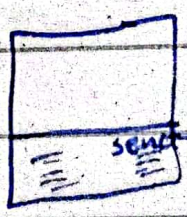
-> Signal R main concept is Hub (Pipeline/channel to make communicati possible).

-> **Hub** : serverside

Methods: ↳ Send Message (string message) ←
↳ Send Notification All

* Clientside Part:

↳ Receive Message



: Client side calls Hub method

: Invoke is used to call Hub Method.

-> Invoke (Send Message, Text)

* We call sendMessage, then sendMessage call receiveMessage to client by "On" metho (Two-way communication)

using Microsoft.AspNetCore.SignalR;

⊙ Steps :

→ HUB class

New Folder (Hubs) →

class NotificationHub

```
public class NotificationHub : Hub
{
    public async Task SendNotification()
    {
        await Clients.Others ...
    }

    public async Task SendNotification2(string product)
    {
        await.Clients.others ....

        await.Clients.All.SendAsync("ReceiveNotification",
            $"{product} has been purchased by someone else", "some other data");
    }
}
```

se

:(functionName, data

→ connection.on("ReceiveNotification", function(Notification, data)

```
{
    cousole.log(notification + ":" + data);
}
```

→ builder.Services.AddSignalR();

→ app.MapHub<ChatHub>("/chatHub");
app.MapHub<NotificationHub>("/nHub"); (Any string)

→ connection Builder().withUrl("/xyz", "/chatHub")
.build();

12/2/25

→ Implementation of SignalR:

(i) MVC Project → Client Side Library ← Dialog-Box (UNPKG) ← Search SignalR in Library (Microsoft SignalR)

↳ Choose Specific file → Dist → Browser → SignalR.js / SignalR.min.js → Target location (www root)

↳ Install

→ Hubs Folder in solution → Class → Notification Hub → Add Library : SignalR → Public Async Task SendNotification()

{ await Clients.All.SendAsync("ReceiveNotification");

→ Program.cs:

```
builder.Services.AddSignalR();
app.MapHub<NotificationHub>
        ("/notifyHub");
```

⊙ Home → Views → Index.cshtml:

```
<div class="row p-1">
    <div class="col-6 text-end">
        <input type="button"
            id=
```

⊙ site.js:

```
document.getElementById("sendButton").
    addEventListener("click",
        function (event) {
            connection.invoke("SendNotification");
            event.preventDefault();
        });
```

⊙ Program.cs:

site.js:

```
var connection = new SignalR.
    HubConnectionBuilder().withUrl
    ("/xyz").build();

document.getElementById("sendButton").
                disabled = true;

connection.on("ReceivedNotification",
    function () {
        console.log("signalR works");
    });
```

→ .layout.cshtml:

⊙ `<script src="../lib/microsoft signalr/dist/browser/signalr.js"></script>`

⊙ connection.start();

⊙ public class ChatHub:Hub

{

public async Task SendMessage(string data)

{

await Clients.All.SendAsync("ReceiveMessage",
"sports week is coming ahead");

}

→ For chat:

```
concurrent Dictionary<string,string>
                    Users= new

public async Task Register
                    (string username)
{   Users [Context.ConnectionId]
                    = username;
    await Clients.Client(Context.
                    ConnectionId).
    SendAsync("ReceiveMessage",
                    "System", $"You
    are registered
    as {username}");
}
```

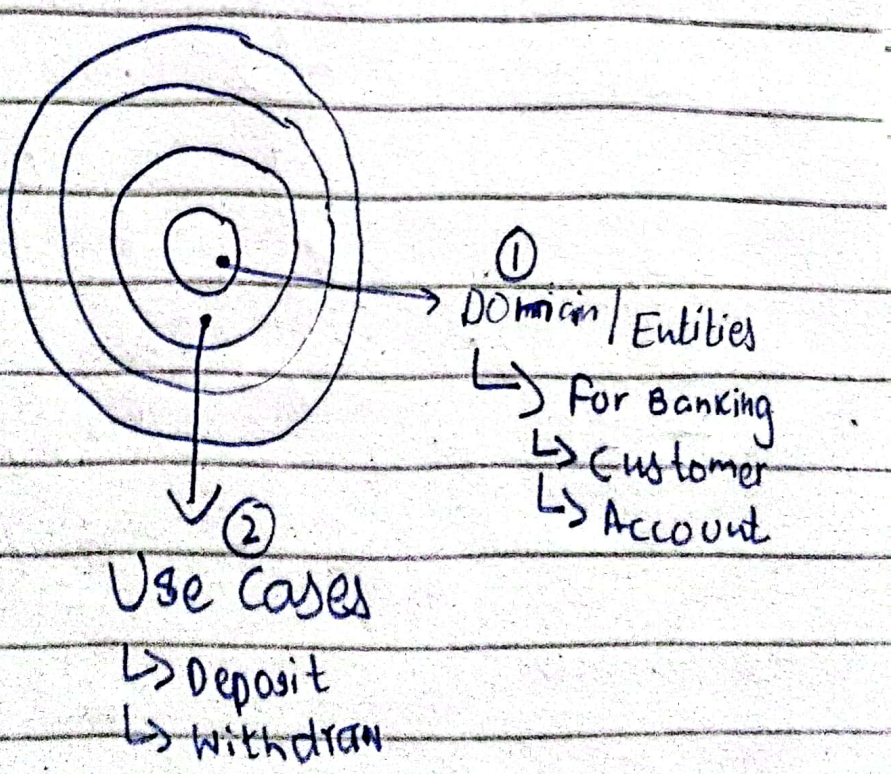
② CLEAN ARCHITECTURE :

→ Our Application mainly depend on technology.

→ Before webforms are used instead of MVC.

→ If technology obselete we hare to convert it into new technology (language).

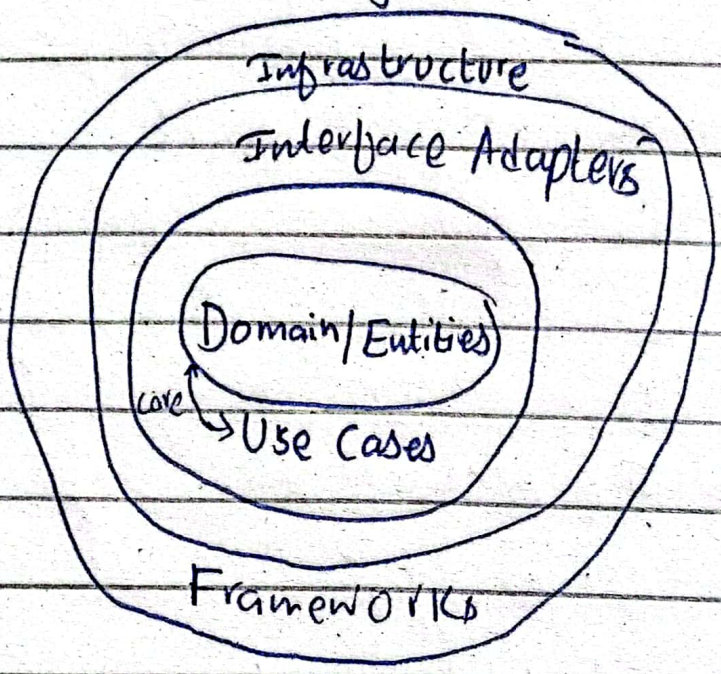
→ This architecture is also called Onion Architecture.



19/2/25

⊙ CLEAN ARCHITECTURE :

→ It is used to remove technology dependency.



⇒ Make class libraries for different layers

⇒ In MVC, Controller act as Interface Adapters.

→ Projects : Entities, Use cases, Infrastructure, frameworks.

→ Implementation of code:

(*) Make Blank solution → Add New Project → Class Library → Domain

* Add → New Project → Class Library
Application ↵

* " " " "
Infrastructure ↵

* Add → New Project → MVC
Web ↵

⊙ You can delete class1 ....

⊙ Domain:

→ Folder (Entities) → Add class
Product

```
class Product
{
    public int id {get; set;}
    " " string Name " "
    " " decimal price " "
}
```

⊙ Interfaces — Entities
Application

→ Folder (Interfaces) → Add
Interface
(IProduct
Repository)

```
: public void Add(Product p);
                Update
                Delete
                Get By Id
IEnumerable<Products> GetAll();
```

⊙ Models → Business layer
→ Frontend layer (MVC)

④ Application :

→ Add Folder (Use cases)

Add class (GetAllProductUseCase)

Different for each use case

Add Reference of Domain

```
:
    private readonly IProductRepository
                    _respository;
.... (constructor) (IProductRepository
                    respository)
    {
        _respository= repository;
    }
    public IEnumerable<Product> Execute
                                ()
    {
        return _repository.GetAll();
    }
```

⊙ For AddProduct :

→ Add class (AddProductUseCase)

```
: private readonly IProductRepo...
                    _repository;
```

.... (constructor) (IProductRepository repository)

{ _respository = repository;

}

public void AddProduct(Product p)

{

_repository.AddProduct(p);

}

* Infrastructure:

→ Add class (Product Repository)

: Add Reference of Domain & Application

: class ProductRepository : IProduct Repository

{

public IEnumerable<Product> GetAll()

{

// Get product from database

(ADO.net , EF-Core etc...

: Add DbContext file in it

List<Product> products = new ...

products.add(new Product

{ Id=1, Name= "Product1" price = 22.34 }

" " "

://context.Products.ToList();

return products;
}

⊛ Frameworks:

→ MVC → Controller (Product Controller)
: Add Reference (Application
: private readonly GetAllProducts
UseCase
- getAllProductsUseCase;
: .....
- getAllProductUseCase = getAll
ProductsUseCase

: public IActionResult Index()
{
Var products = - getAllProducts
UseCase.
Execute();
return View(products)
}

→ View:
Product folder → Index

: Make web as startup project

→ Index.cshtml :

```
@model IEnumerable<Domain.Entities.Product>

<h1>All Products</h1>

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

⊙ program.cs :

```
builder.Services.AddTransient
        <GetAllProductsUseCase>();
builder.Services.AddTransient
        <IProductRepository,
        ProductRepository>();
```

⊛ Before we have done usecase based, now will be Service Based.

→ Application (by service)

→ Service Folder → ProductService Class

: public void Add(...)

public void Update(...)

Delete

GetbyId

: In controller then change Usecase to service.

: C → Application → Domain

: Real logic is in Infrastructure

(₹) Only Web will change if other technology is introduced (API)

(₹) Homework of Clean Architecture Also Add in Project.

→ Clean Architecture | → SOA, → MVC, → Micro Services

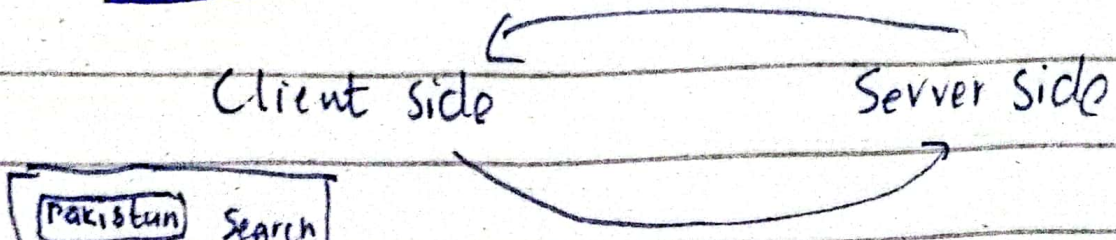
API

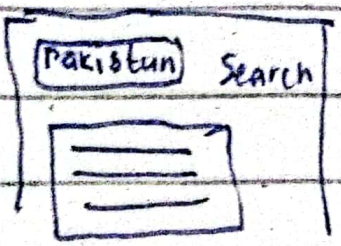
(₹) Clean Architecture in Project

Clean Architecture Project

50%	3rd March
75%	10th March
100%	17th March

AJAX:





→ To update the page partially.

- Client side language Javascript
  - ↳ JQuery
    - ↳ .get
    - ↳ .post

- Code:

```
$(document).ready(function() {
    $("#b1").click(function() {
        var data = $('#txtData').val();
        $.get("/Home/MyAction", { inputData: data
```

```
, function(result) {
    $("#partialPlaceholder").
        html(result).fadeIn
        ('slow');
    });
}
}
```

→ index.cshtml: : AJAX in project

```
<div>
    <input type="text" — —
    — — — —
</div>
```

3/3/25

⊙ Service-Oriented Architecture:

→ Business Logic remains same for other devices code changes.
→ API can be used at multiple places.

→ Purpose is to remove Platform dependency (windows, Android etc....) and language dependency.

→ Services Implementation Types:

(i) Restful Services

(i) GRPS services.

① Restful services:

→ It is identified by two things:

(i) URL (A function is assigned URL)

(ii) HTTP Protocol Methods (GET, PUT, READ etc...)

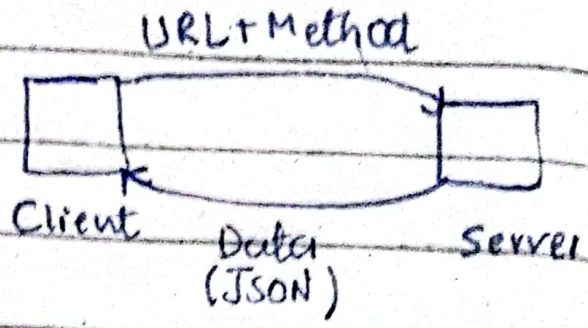
→ A service will return something in form of data (JSON format)

→ Status code (200 - 500)
(Successfully created) (Client-Side Error)

→ Representational State Transfer (State is transferred in representable form)
(Architecture Model).

→ HTTP Method:

C	Post
R	Get
U	Put
D	Delete



Implementation:

→ Project → ASP.Net Web API

MyFirstRestfulServices ← .NET 8.0 ← Select Use Controllers

→ WeatherForecastController
(ApiController)

: [Route("api/[controller]")]

: www.xyz.com/api/WeatherForecast

[HttpGet] [HttpGet(Name = "GetWeatherForecast")]

: [Route("MyWeather")]

→ Swagger is used to check or test services

* (PostMan is also used to test services).
(Install PostMan)

→ Get Product ALL :

Controller → API controller

Product Controller ↵

5/3/25

→ Product:

- Id
- Name
- Price

→ Product Repository:

- Add
- Update
- Delete
- ReadAll
- ReadbyID

- Implementation :

→ New Project → ASP.Web API
.Net 8.0

→ Models (Folder) → Product Class

```
Product
{
    Id , Name , price
   (int) (string) (decimal)
}
```

→ Models → New folder (Interfaces + Repositories)

→ Interfaces → New Interface (IProductRepository)

```
IProductRepository
{
    Add(product)
    Update(product)
    Delete "
    List<Product> GetAll();

    Product GetById(int id);
}
```

(*)

```
Add(product)
{
    List<Product> products = new List<Products>;

    products.add(product);
}

Update(product)
{

}
```

→ Repositories → Product Repository

```
ProductRepository: IProduct
                   Repository
{
    List<Products> _products;

    public ProductRepository()
    {
        _products = new List<Product>()
        products.Add(
        new Product(){ Id=1, Name=
                       "Apple",
                       Price=23.45} )
    }

    : GetAll()
    {
        return _products;
    }
```

→ Controller → Add NewItem

Name ← API Controller ←

(ProductController)

→ ProductController

{

private readonly IProduct
Repository
- productRepository;

public ProductControllr(
IProductRepository —
{ - productRepository =
productRepository
}

→ Program.cs

builder.Services.AddTransient<IProductRepository,
ProductRepository>

→ public ActionResult<IEnumerable
<Products>>
GetProducts()

[HttpGet]
[Route("getall
Products")]{

return - productRepository
.GetALL();

}

return Ok(-productRepository.
GetALL());
Explicit

& 500
[Server side Error]

→ without [Route("getallProducts")]
it not returns the route
after Product, but it will
also work.

→ Get By Id:

☐ /api/Product/getbyID/2

[Route("___"/{id})

[HttpGet]
[Route("getbyId/{id}")]
→ public ActionResult<Products>
GetById(int id)

Controller

Exception Handling

try {
{ return _product...
}
catch(exception ex)
{ return
NotFound();
}

return _productRepository.
GetById(id);

Repository: , GetALL()
return _products.where(p=> p.Id=Id).
single();
}

-> Add:

① Repository:

```
public void Add( — )
{ -products.add(p);
}
```

② Controller:

```
[HttpPost]
[Route("addProduct")]
public ActionResult CreateProduct
                    (product)
{ -productRepository.Add(product);
  return OK();
```

return CreatedAtAction (nameof(GetProduct), new{id=product.Id}, product)

: Do input add in Swagger.
(Success Code Check) (Then check GetALL())

-> Program.cs

Instead of Transient we use:
(New object each time)

.... Add Singleton .....

: 201 (created code) (Product return which is created Now)

→ Update (int id, Product Updated Product)

10/3/25

→ Quiz : Clean Architecture, SignalR, N-tier and Web API (Quiz in Next Class) (Vivas on Thursday)

- 200 → OK 404 → Not found
  201 → created

→ **Route:**

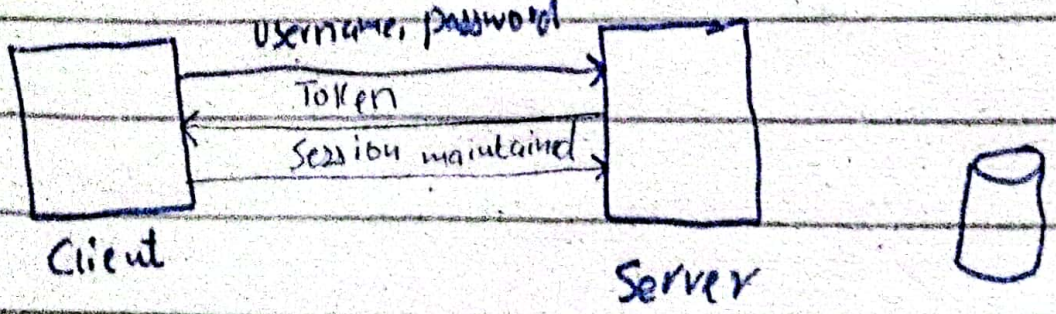
: methods (i) [HttpGet]
[Route("secure-data")]

(ii) [HttpGet("secure-data")]

: [HttpGet("secure-data/{id}", Name="MySecureData")]

Name = "" (Meaningful Name of API)

→ **Secure:**



(*) **Token** based Authentication is used in Web API to make it secure. (JWT Token) Request Generated and Token is also sent and received on client or server side.

-> Three parts of JWT Token:

(i) Header Part

(ii) Claims Part

(*) Implementation:

=> API Project

① -> Package install ("Microsoft.AspNetCore.Authentication.JetBearer") (8.0 Version)

② -> Program.cs:

```
builder.services.AddAuthentication
(JetBearerDefaults.
Authentication.
AddJwtBearer(options =>
options.TokenValidation
Parameters ....
```

(From Git Project)

: Lecture -2
(API - Security)
(GitHub):

: Login Model
{ Username
, Password
}

: Secret-Key from token.

③ → Controller (Auth Controller)

Login (Action Method)

[HttpPost ("Login")]

IActionResult Login ([FromBody] LoginModel model)

{ if (model.Username == "user" pp

. . . . . |

: Policies

Token Generation

:

return OK (new { token = new JetSecurity TokenHandler().Write

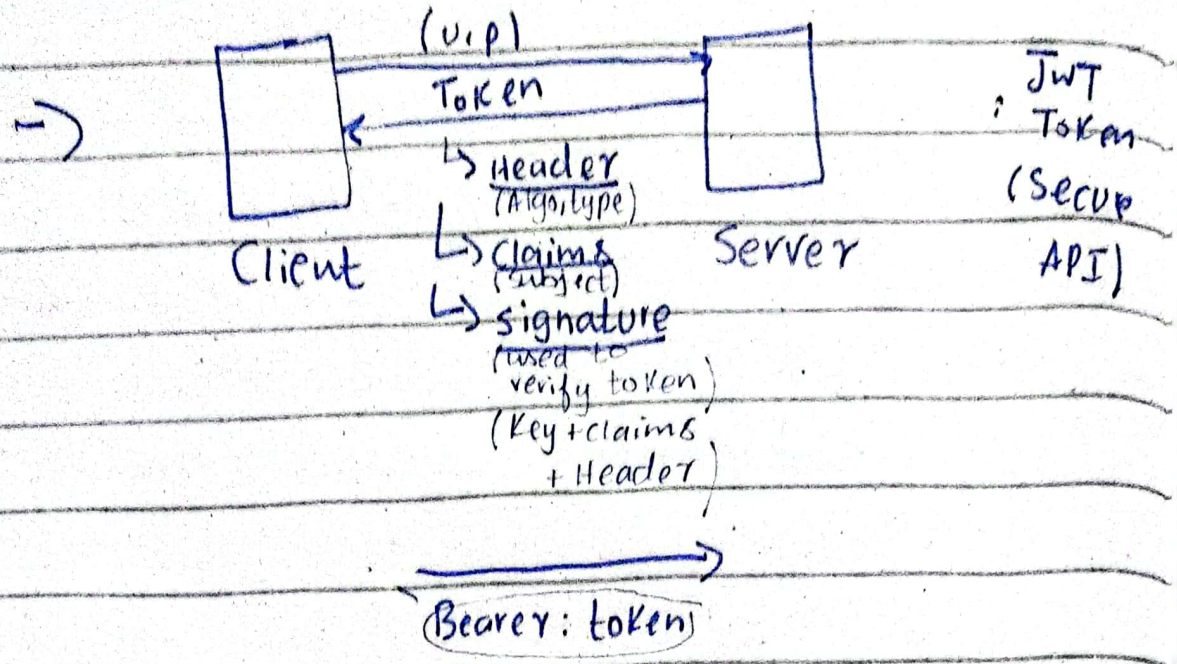
}

→ Token is not sent by swagger.
So we use PostMan.

→ PostMan → New → HTTP → Authorization

"This is protected data" ← Token ←

12/3/25



→ Token compare for valid request

* ABCD (Request Token) = ABCD (Valid Token) — Server Token

* DEFG ≠ ABCD (Not Valid request) — : Secret Key expose API's

→ Header and Claims can be decoded and signature is irreversible. (JWT Online Decode)

* Authenication of API is done, now authorization is done by policies.

→ builder.Services.AddAuthorization
(options =>
{ options.AddPolicy("EmployeePolicy",
policy =>
policy.RequireClaim("Department",
"HR"))
});

→ RestrictedController.cs:

[Authorize(Policy="EmployeePolicy")]
[HttpGet("restricted-area")]

→ Login Action:

if(...)
var claims=new[]
{ new Claim(ClaimTypes.Name,
model.Username),
new Claim(ClaimTypes.Role,"Admin"),
new Claim("Department","HR")
};

→ Degree SE Reserve:

```
builder.Services.AddAuthorization
                (options =>
{ options.AddPolicy("DegreePolicy",
                    policy =>
    policy.RequireClaim("Degree", "SE")
});
```

: Login Action Method:

```
:
Var claims = new[]
{
    new Claim(ClaimTypes.Name,
              model.Username),
    new Claim(ClaimTypes.Role,
              "Degree"),
    new Claim("Department", "SE"
              Degree
};
```

: StudentController:

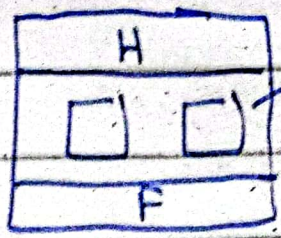
```
[Authorize(Policy = "DegreePolicy")]
```

```
[Http Get ("restricted-area")]
public IActionResult Login()
{
    return Ok("Access granted...");
}
```

: Lecture on GitHub (web API)

⊕ Asp .Net Core:

- MVC
- SignalR
- Web-API
- SPA
  - Frameworks → JS (React, Angular, View, Next)
  - → Blazor

→ 

This page is made by 'compents'

: Components is building block of Single Page Application (Blazor Pages)

* Next Week Quiz (Monday and Wednesday)
Viva will be be on next week

17/3/25

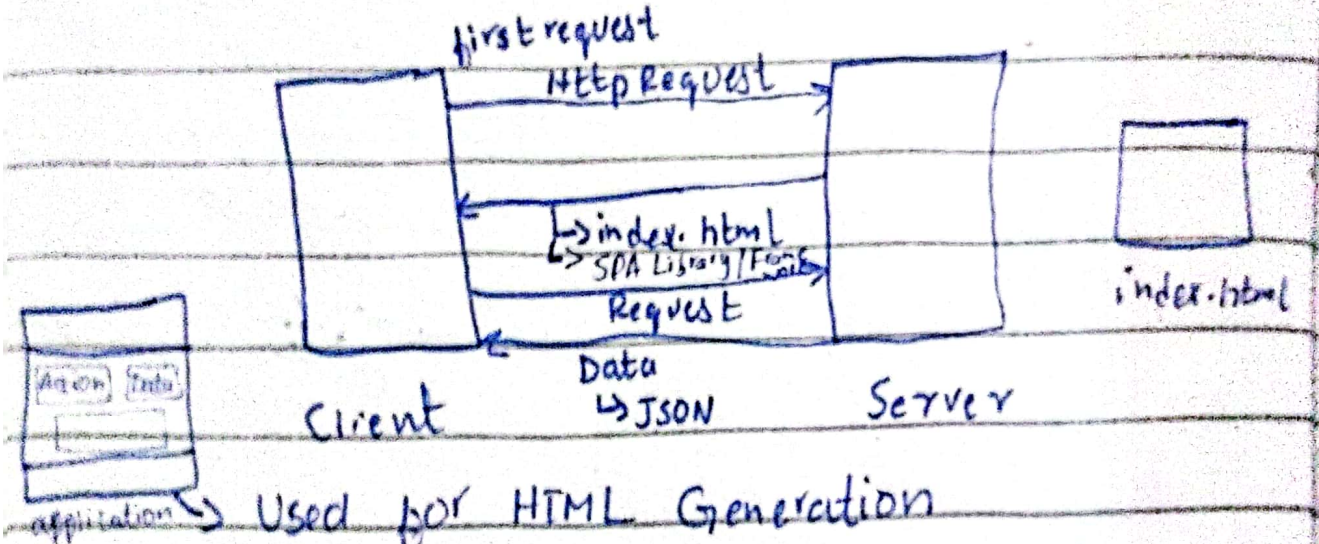
* SPA (Blazor):

→ Single Page Application

↓

One page

(All libraries and data and view will be created on client-end)



→ Used for HTML Generation

→ C# is used for both backend and frontend and on both sides client and server.

→ Browser can read HTML, now browser can't read C#, so converter is used on client side which is "Web Assembly"

→ Web Assembly is intrepreter which provide facility to execute high-level language on web.

→ dotnet runtime is downloaded on client side (dotnet.wasm)

→ Downloadable on client-side.
- dotnet runtime (dotnet.wasm)
- application
  - ↳ dependencies

→ For getting data web API's are used.

→ Blazor: (Types)
(i) WebAssembly

Implementation:

→ Template search → Blazor (Web Assembly Stand Alone App)

Latest .Net 9.0 ← My First Blazor App ←

↳ Create

→ Folders → Layout
↳ Pages

→ Page → Index.html
: Everything is loaded in div app
< div id = "app" >

→ Program.cs:
: We will everything in SPA as component.

: Component is basic building block.

→ pages → counter page: (component)

@page "/counter"

(*) Home component:

@page "/"

(*) Weather component:

@page "/weather"

inject :(DI)

→ Component three parts:

(i) Directives (iii) C# code

(ii) Html

→ Run → Inspect → Application → storage

Reload ← Network ← clear ←

(All files are downloading again).

(main, Project Name files)

→ Component which contains button and on clicking it displays button is clicked

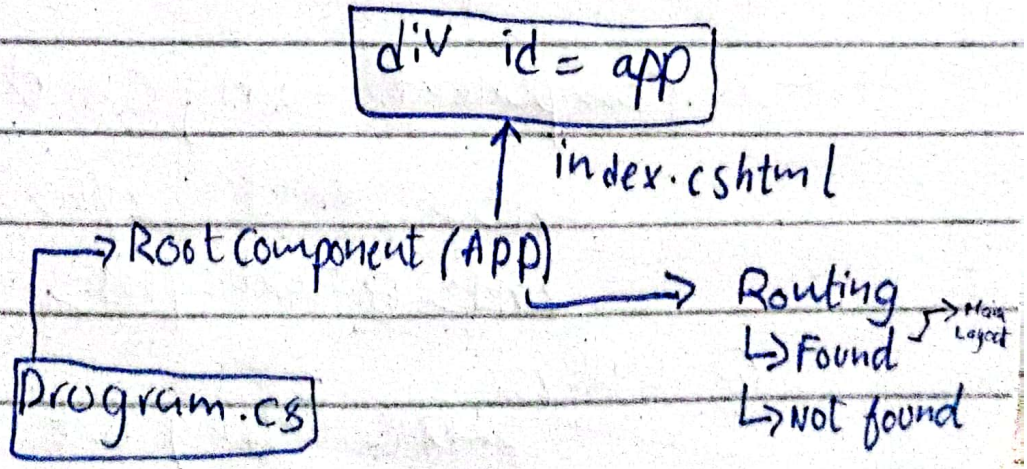
```
@page "/buttonclick"

<h1> Text </h1>
<button value="change" @onclick
                        > changing text >

@code
{ string text = "default text"

    static void changingtext (){
        text = " Button is clicked";
    }
};
```

19/3/25

Parameter passing → HttpClient Method for Requests.

→ Weather Request verification from Inspect.

→



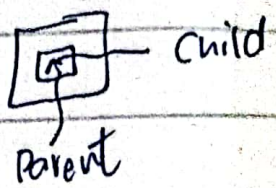
→ weather
→ Home
→ Counter

→ MainLayout → @Body()

→ Component
↳ Directives
↳ Page
↳ To define route

↳ Markup
↳ Html + C#

↳ Code
↳ C#



Implementation

→ Pages → Component (My Parent Component)

: Parent to child

@page "/mycomponent"

<mychild component>
</mychild component>

<h3> My Parent Component </h3>

@code{
string data = "This is some data to send";
}

→ Component → My Child Component.

<h3> </h3>

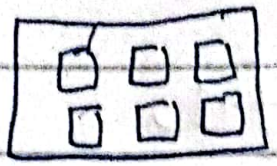
<h1> @Text </h1>

@code{
public string Text {get; set;}
= "This is default text";
}

My Parent Component:

```
@ for each (var p in products)
{   <MyChildComponent Text = "data"
                      Name = @p.Name"
                      price="@p.Price">
                      </MyChildComponent
```

```
@ code
{   string data = "this is some data to
                   send"
    List<Products> prod = new List<Product>
}
```



(Product cards (Component same) but data changes (text changes) for each card).

→ Child component:

```
<h1> @Name</h1>
     @ Price                    :<a href="/data(
```

```
@ code
{   [Parameter]
    string Name {get; set;} = " ";
    [Parameter]
    string Price {get; set;} = " ";
}
```

: Wednesday (Two lectures of MAD)
: Monday (Two lectures of SCD)

→ For Routing:

: Counter Component:

```
@ page "/counter/{Id}"
<Page Title> Counter </Page Title>
<h1> @ Id
<h1> @counter</h1>
```

· Make Id in Child

: Counter : public string id

→ @ page "/counter/{Id:int};
@ page "/counter"

: Make id as int now

: Delete .Data if Issue error

→ public int id {get; set;} = 34

Also add Name:

→ [Parameter]
public string Name {get; set;}

@ page "/counter/{Id:int}/{name}"

24/3/25

① Blazor App (1-3 Group members)
(Make Front-end before mid)
(1- Page Description till Friday)

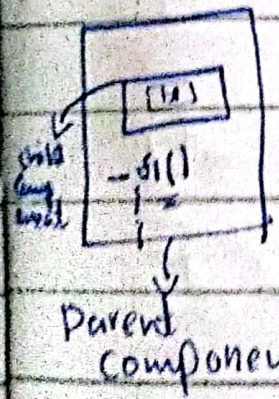
② Microservices, Restful Services (Graph etc ...), SPA.

⊙ Blazor (SPA):

→ Pages is replaced with components (Building block).

↳ **EVENT CALL BACK**

⇒ By clicking on button, we wanted to call method. Button will be in child component and method in parent component, so we wanted to link them.



⇒ When we wanted to execute parent component method by clicking child component button.

→ Implementation:

① Parent component and
Child component (Create it)

→ Child:

<button> Child Component
Button </button>

→ Parent:

<h1>@Data</h1>
@ code
{
public string Data {get; set;}

public void MyMethod(){
Data = "Child component
button is clicked";

→ Child: [parameter]

@ code → public EventCallback
onChildCallback
{get; set;}
{ private void ChildMethod()
(CallMe)
{
}

(callMe)

< button @onclick="childMethod">
childComponent Button</button>

: childMethod()
(callMe)
{
onChildCallBack.InvokeAsync();
}

-> Parent:

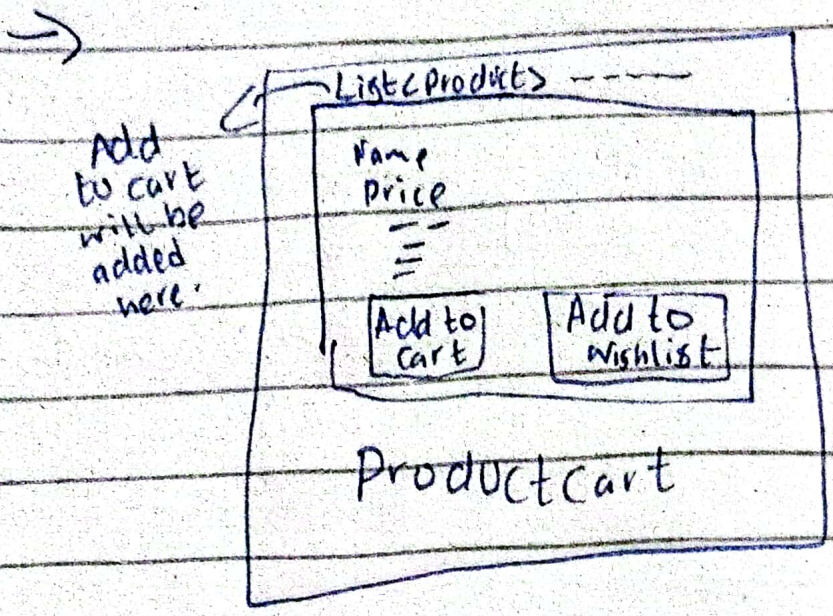
<h1> @Data </h1>

< CC @OnChildCallBack="MyMethod" >
(childComponent) </CC>

-> Now we wanted to send from child to Parent:

: Child -> public string childData {get; set;}
private void CallMe()
{
...
onChildCallBack.InvokeAsync
(childData);
}

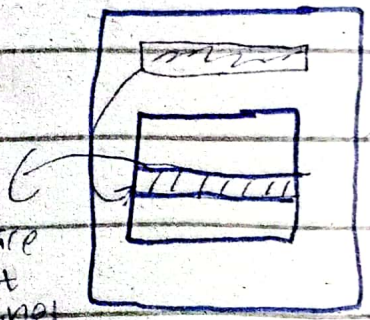
: Parent
@code
{
MyMethod(string x)
{
Data = " " +x;



⊙ Templated Components:

⇒ Child UI is fetched from parent component

↳ Render Fragment



: After Mid (we will discuss it)