1-2-23

arameter. action = Point to the Url.
↳ bydefault we have Relative URL.
"javascript : void(0)" ⇒ Page will not be reloaded.
or
"#" ⇒ Also ~~not~~ not reload the page.

→ Script can be written before HTML if it has JQuery.

⇒ 2ways to create button/(node)
(i) Simple element :- CreateElement
(ii) Nested element :- cloneNode
↓
with true value it will create nested element.

→ Passing inside backtic(``) → it will render dynamically.

⇒ fetch(Url,{
method : 'GET/POST/..', headers: {'Accept':''
'Content-Type':..
},
body : {..}
}).then (callback);

↳ content-type is mandatory.
↳ when service is returning some data & that is processed by the the client, then ~~it~~ Accept is mandatory.

Page
Date

base class for serializer.

→ Media-type formatter :- To send custom form data to the client from server.

→ serializable ⇒ converting data into standard format
↓
(can applied on any datatype).

→ To prevent from fetching the unnecessary refetch, can use paging on server / client side. ↳ fetching / limit size in query.

⇒ To changing the default behavior :
↳ place against the parameter [FromUri and [FromBody]
default behavior :- { Header : primitive types
{ Body : complex "
↳ expected by API.

* 2-2-23 :

→ Automapper :-
↳ Convert model to view model
→ flating of the object
↳ complex obj has Nested obj
Instead Make Simple object from it.

① → Provide Mapping in ProfileClass:-
CreateMap < source, dest^{n} > ();

→ GET Req:- Convert (model) data obj to view format,
TodoItem — TodoItemDTO.
fetched from DB — viewed by user

② Add dependency Injection in Program.cs file.

↳ To use mapper in any class use "IMapper" as it is provided by Automapper to refer to our mapper.

→ _mapper.Map<TodoItemDTO>(todoItem);
datatype of desired obj. , current obj

⇒ For put and post request :- "ReverseMap()" is required.
↓
This will perform both orignal mapping (Item → DTO) & reverse also (DTO → Item).

copy to todoIte

Put req:- _mapper.Map<TodoItemDTO, TodoItem>(todoDTO, todoItem);
Referencing to our one of the item of DB list.

Page
Date

→ In get req. :- new object get's created in In memory.

I.P :- CreatedAction (Url, param, res. body data)

⇒ To Mapp different name field's of sour→des
↳ .ForMember (dest ⇒ dest. TaskName ,
opt ⇒ opt. MapFrom (src ⇒ src.Name)).

I.P :- Identity ⇒ means Auto Increment.
↳ Also try to handle database Exception.
↳ Some default computation ~~et~~ can be written in client / server -side / in DB in terms of trigger.

→ change default behavior of Mapping fields to $dest^n$:-
.MapForm() , .Ignore() , .NullSubstitude()
use in:-
↳ raw data conversion to obj ⇒ when DB is not supporting nested obj.

Page
Date

controller should do only one work

S → single responsibility
O →
L
I
D

→ SOLID principle :- separation of concerns.

→ IMapper :- to convert DTO to View & view to DTO

→ DI is required for all different type of service class because all different type of DB connection uses different library.
ex :- "limit" supported by MongoDB, but not by all.

→ SQL is ISO standardized.

* 16-2-23 :

Authentication :- (i) cookie (storage shared betⁿ website)
(ii) token
(iii) local storage
↳ Not shared betⁿ website)

Page
Date

⇒ 15-2-23 :-

* Service lifetime :-

1) Transient :-
→ The object is created every time it is requested from the DI Container.

2) Scoped : The instance will be created once per request lifecycle.

Only One Instance for both middleware

3) Singleton :- Only one Instance for the entire life cycle of the application.

→ GetGuid() :- To Create Unique Id.
(global)

→ Transient :- t1 & t2 both point to different object.
(reference)

scoped :- sc1 & sc2 both point to same Instance (object)

singleton :- sn1 & sn2 both same throughout the lifecycle of application.

→ whenever req. from client to API is done, then token is required to pass.
Ex:- login via google account, facebook account, etc.

↳ token passed in req. header & passed to oauth server to validate client.
↳ API can directly verify, itself.

→ JWT ⇒ Jason web token

→ Along with password we add the key to reduce chances of hacker the account because of same password as it ~~can~~ ~~g~~ will generate different hash key.

→ Token :- Minimum length must be 16 characters.

(i) IAuthRepo → Register()
→ login()
→ UserExists()

$$\begin{tikzcd} & \text{Register()} \\ \text{IAuthRepo} \arrow[ur] \arrow[r] \arrow[dr] & \text{login()} \\ & \text{UserExists()} \end{tikzcd}$$

password
↓ generate
passwordsalt

↳ passwordHash = Combine(password, passwordsalt).

→ ILogger :- provided by the framework.

→ out :- not need to declared it by default available outside f^n / $in f^n$ scope itself.

V.I :- Mostly ↓(by default) cryptography algo. works with bytes.

Page
Date

→ There is a standard set of claims which are created at the time of token creation and custom claim can also be created.
↳ claims ⇒ who has created?, etc.
↳ Payload

→ Token :- subject : claim
expires : How much time token will be valid?
SigningCredentials : Encrypting algorith

→ JwtSecurityTokenHandler() ⇒ Create token.

→ To give only access to some type of user only. (of API)
[Authorize] ⇒ required token to call

↳ Token Value provided :- $\underbrace{\text{bearer}}$, token value
(tell which type of token)

Provide token ⇒ ① Auth, ② Header

→ [AllowAnonymous] ⇒ Allowed to req. by anyone w/o Authentication

22-2-23

→ Application types :-

(i) Monolith Type: All files are linked in one file.

Linux: Kernel is at ⇒ /booth folder
kernel ⇒ 90% code c language
10% code assembly lan.

⇒ cons :- debugging is difficult as all files are combined so where is the problem is hard to Identify.

Y.F: CI/CD :- Automation tools for testing.

→ scaling :- Increase Infrastructure / Resources
↳ On this type of app. ⇒ vertical scaling can be performe
⇒ horizontal scaling :→ Increase Instances of the server.

→ Performance :
↳ Modification is very costly to latest technology (language).

(ii) Microservices :
→ Each service is Independently execute.
→ Characteristics :-
(i) ⇒ Independently deployable

Page
Date

↳ changed from SOA to Microservices.

→ latency ⇒ time for responding of request
→ If database are shared then merge that services as duplication / redundancy should not be there.

→ As far As possible database are not shared
→ Blob store :- specifically optimized for Image storage.

Pros:- Failure doesn't cascade.

→ In this scaling is very easy by specifying set of rules on the cloud.

* Enterprise Bus service :- Multiple services are connected & communicate via this bus.

⇒ 23-2-23:-
* Cross-cutting concerns:-
↳ logging

(i) 3) → API gateway :- do Authentication & authorization Instead of duplicating at each microservices
↳ single entry point
↳ Protocol translation (Ex: Communicate with IOT device)

ngx :- (?)

AMQP : Protocol.
→ API gateway provides Inter operability.

⇒ Use of Proxy server:
  (i) For security (i.e. Authorized user can only access)
  (ii) Gateway

⇒ search :- FACATE Design pattern

⇒ Rate limiting (throttling) :- don't want backend services to be highly loaded.

* Another Design pattern:-

(ii) ~~(i)~~ Backends for frontends

→ For $\overset{(In\ C)}{RPC}$ & $\overset{(In\ Java)}{RMI}$:- both client & server must be written in same language.
  gRPC ⇒ google RPC.
  ↳ Targetting RPC call to be an interoperable.

I.E:- Ocelot :- used as API gateway.
  ↳ freely available
  ↳ ocelot cache Manager can also be used.

$$\text{client} \xrightarrow[\text{Req.}]{\text{upstream}} \text{API gateway} \xrightarrow[\text{stream}]{\text{down}} \text{Microservice.}$$

launchsetting.json
↳ Take ~~[illegible]~~ no. of https

⇒ To get URL of API Gateway :- got from launsetting.Json.
http:80 , https : 443

→ ocelot.Json :- This file has Url translation for API gateway.

→ Data seeding :- Initialization of database table while creating of Model using ModelCreating() method.

* 11-3-23 :
⇒ option :-
-a :- listing all container running (up) & exited (down).

⇒ Entire Image can be pushed onto docker & it can be refered by other team member.

⇒ Change the name of docker hub :
"docker tag oldname newname".

⇒ It will optimize all dependencies modularly (layer by layer).
Ex: base os
↓
node js
↓
~~some library installed~~

windows : .so (shared object)
linux : /user/lib

→ when user run the code
→ In exe file :- static link library.
→ dynamic link library : taken care by os, and it's not part of the code.
  ↳ And this libraries are dynamically linked.

Ex :- static library : stdio, printf, scanf, sqrt.
  ↳ part of executable file.

⇒ While Pushing the docker Image
  ↳ If layer already exist, then it will not download it (push it).

→ -d : detach (run in background)
→ -p : Port mapping.
→ -q : to list down only container id.
→ -a : all container (running & not running)
  ↳ otherwise only "ps" only running container to list down.

→ -it : interactive terminal.
→ To go inside docker :-

$$\text{\$ docker } \underbrace{\text{exec}}_{\text{go inside container}} \text{ -it demo } \overset{\text{Process Status}}{\text{ps}} \ \underbrace{\text{aux}}_{\text{(all users)}} \quad \text{↳ list down all process}$$

Vm => marden edito

H.W :- FTP, telnet , ssh

Page
Date

→ /bin/sh :- open the shell
  ↳ opening inside conteines
    ↳ If user wants to change some files the do using this.

Ex :- can change index.js using any editor by opening it on shell.

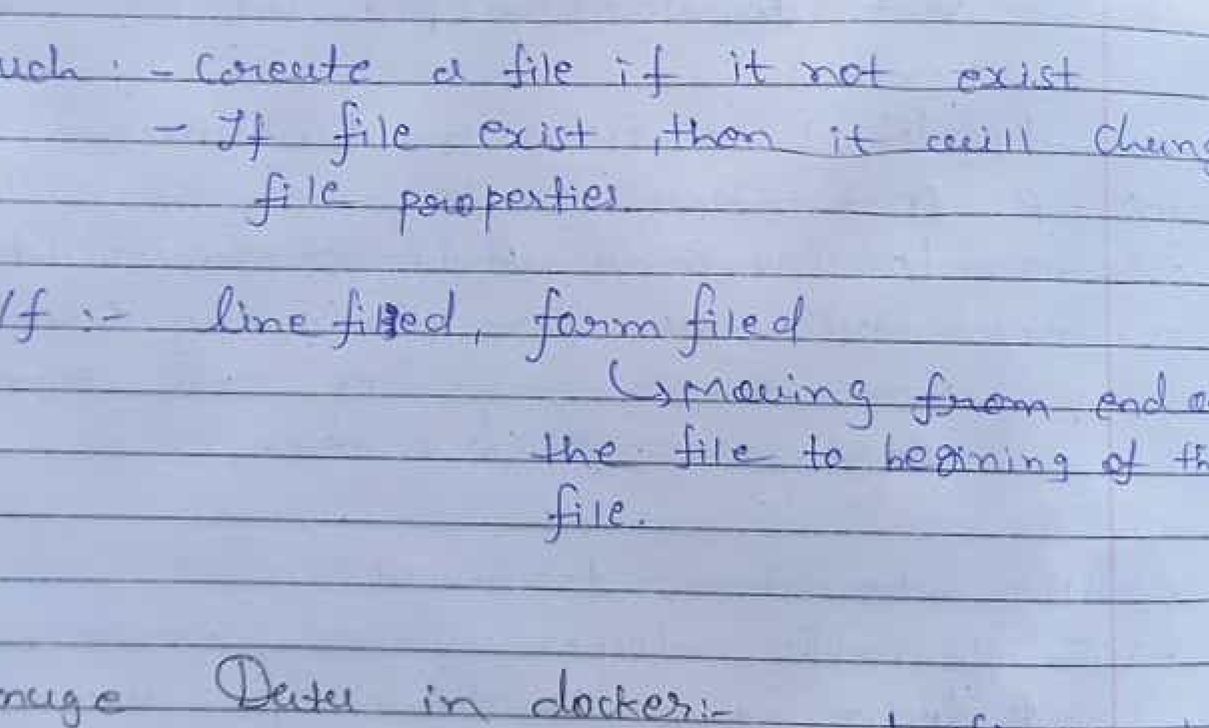
→ ssh :- generate SSL certificate & it can be secu led
  ↳ Hence it is secure.

⇒ To save changes made in container:
  ↳ commit the changes.
  " docker commit containerId "

→ touch :- create a file if it not exist
  - If file exist, then it will change file properties.

→ \r/f :- line filed, form filed
  ↳ moving ~~from~~ end of the file to begining of the file.

* Manage Data in docker:-



① bind mount , ② volume:

→ Now ~~on~~ our data is stored in some volumn , which can be used by other.

↳ Persist the data even if container exit.

* 15-3-23 :-

→ Exe file verified by using checksum file, which Prevents user to download exe file given by Hacker (Malicious user)

→ Everytime we can't create new version of Image for each change.

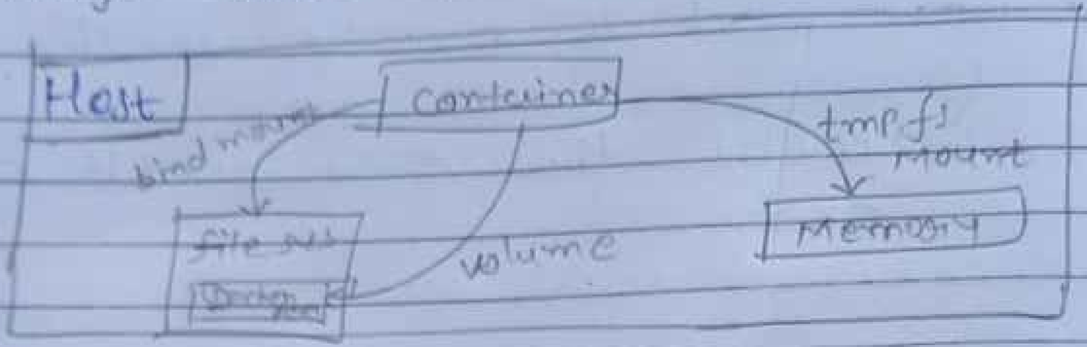
Solⁿ: ↓ volume.

⇒ Host OS : Required to Interact with Hard dist for saving data.

guest OS : storing Modules, running apps like Angular, NodeJs it is required.

NFS = (9)
(n/w file system)

→ Manage data in Docker:-



⇒ File system:
On windows :- FAT, NTFS file system

On Linux :- ext2

↳ Because using filesystem Mem. access is done by OS.

→ whenever new storage (Media) is inserted, then binding to new storage is done by file system.
↳ This binding called mount.

→ -v :- "volume : directory"
↳ for volume
-e :- environment variable.

I.E :- Bydefault Mysql doesn't allow login remotely.
For remote login : "username @ ip addr" (host name)

Try to run on cloud AWS, GCP

★ CNCF (cloud native ~~computing~~ computing foundation)
↳ support docker & Kubernetes.

→ "--network" :- To run on same m/c Two apps.

→ "--network-alias" :- can refer container by this name Instead of IP address.

demo-mysql-2
↳ root folder

★ Kubernetes :-

→ orchestration to manage the container.

→ cluster = collection of machines.

→ rollout :- when admin creates one or more containers.

rollback :- stopping of 1/more container

→ SSH key :- Remote login to another machine securely.

→ can link to your cloud & or local machine

→ pod :- group of containers.
Ex: node-app + Mysql
2 container

Page
Date

→ Kubernets :
  used for production ⇒ on cloud

※ docker can manage "containerd" type of containers.

⇒ cubectl → need to enable it before docker cubectl run, otherwise last at cubectl will not run.

port : outside : Inside container

→ primise cluster : local cluster
  cloud clusters