Express
js

Library V/s framework

Library → library किसी code के छोटे पार्ट को control करता है या किसी code पार्ट को complete करता है।

Library मे हम control करते हैं कहाँ लिखना है और किस तरह से लिखना है। eg. axios

Framework → framework (pre-written) कोड लिखता है और हम उसे use करते हैं।

→ framework मे काफी सारा control उसका ही होता है कि वह किस तरह से लिखेगा।

→ It used for server side programing.

→ यह incoming request को सुनता है कि कब request आएगा।

→ request को pass करता है Node js को।

→ Match response with routs

→ sutable request को response करना।

www.google/search/

eg - express

Server listen for request

→ Server request के लिए wait करता है और जब उसको request मिलता है तो Port के अनुसार उसको phir response करता है

step ①

Make a different folder and create a package.json

step ②

go to npm website and write "/express" in search then write the "npm install express" in terminal.

step ③

make a index.js file in the folder

index.js

```
Const express = require("express");
  Const app = express();
  let port = 3000;
  app.listen (port, () => {
      Console.log("app is listening on port ${port}");
});
```

Step (9)

Write in terminal node index.js

↓

Now the express ready to listen request

↓

then go live the page

↓

in the search bar [Q \3000] → port

write the port number "8000" then page is runing

↓

also you can write "localhost:3000" its also work

Note

→ index.js को Run करते ही server start हो जाता है और listing the request

→ port का मतलब होता है कि Connect करने मे मदद करता है web server और web Client के बीच

→ to stop the server "ctr + c" in terminal

Sending request to server

```
app.use((req, res) => {
    // Message or Code
});
```

previous step run

index.js

```
Const express = require("express");
Const app = express();
let Port = 3000;
app.listen(Port, () => {} // request listen
    Console.log("Server listeing for request");
});
app.use((req, res) => {} // request reive
    Console.log("request reived");
});
```

next Step

Run "node index.js" in Cmd terminal in VS Code

Next Step

Now server ready to listen request

↓

Go to broswer

↓

type in search bar "localhost : portNumber" → 3000

↓

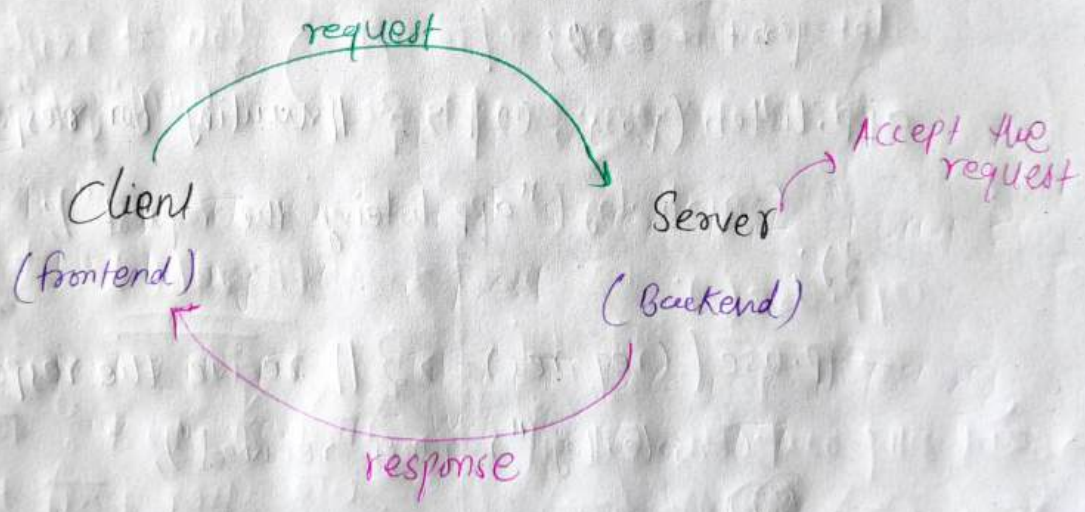
then the broweser look like :) bubbling

↓

browser send the request to server

↓

Check vs code the request received!



Sending response

→ जब आप server को request भेजते हो तो वो आपको response मे वही भेजेगा कि जो आप Port query search को match करता है।

like – Search मे Port Number जो होगा वही request का response send होता है

Previous Code index.js

```
Const express = require("express");
  Const app = express();
  let port = 3000;
  app.listen(port, () => { // waiting for response
      Console.log("app listeing the request");
  });
  app.use((req, res) => { // recive the response
      Console.log("request recived");

      res.send("Hello World"); //send the response
  });
```

next Step

Run the "node index.js in Cmd terminal

Next Step

Open browser
↓
search "localhost:portNumber" → 3000
↓
then the response show on the webpage

Note

→ एक time पे एक ही response को send कर सकते हैं

express js response method

① res.send() — Sending response for (string, JSON, bubber, Object)

```
res.send("Hello");
```

② res.json() — When you are working on API or need to send the JSON data then this method helpfull.

```
res.json({message: "Data send", status: 200});
```

③ res.status() — Necessary to handle different type of HTTP status code (eg. 200 for sucess, 404 for not found)

```
res.status(404).send("Page Not found");
```

④ res.render() — It using template engine rending dynamic

(5) res.redirect() – Usefull for redirect the URL to another tab, such as the submission or login

```
res.redirect('https://www.google.com');
```

Routing

→ इसमे ये different rout पर different response send करता है

like →

www.amazon.com/phone
www.amazon.com/search
www.amazon.com/Home

} different rout मे ये different response भेजता है using "get"

app.use()

→ इसमे आप rout पर कुछ भी गलत लिखो फिर भी आपको response मे कुछ न कुछ मिलता है अगर response भेजा जा रहा होगा। जो कि सही use नही है ☒

app.get() best

→ लेकिन ये best है क्योंकि जब user किसी button पे click करता है तो URL मे clicked button route add होता है और clicked button name "/match" होता है और response show होता है ☑

previous code edit

one path → one response

index.js

```
Const express = require("express");
  Const app = express();
  let port = 3000;
app.listen(port, () => {
    console.log("Server listeing the request");
  });
app.get("/", (req, res) => {
    res.send("you conneted with root path");
  });
app.get("/search", (req, res) => {
    res.send("you can search anything");
  });
app.get("/product", (req, res) => {
    res.redirect('link');
  });
```

nextstep

Open vs code terminal and write "node index.js"

pto →

Next Step

Serve is now listen request

↓

go to browser

↓

Search the "localhost:3003" Enter

↓

Server catch the request

↓

write "/search" in URL Enter

↓

again write inplace of search to "/product"

↓

Each route for each response in expressjs

Note

अगर user कोई गलत rout request send कर दो तो उस समय आप अपना custom error send कर सकते है। सबसे नीचे लिख देना।

```
app.get("*", (req, res) => {
    res.send("sorry its not working");
});
```

Note इससे error नही आएगा कि Custom message आएगा जो कि 404 से बढ़िया लगता है।

nodemon

→ nodemon एक Npm package है जो कि एक service देता है जब आप file मे कोई change करते है तो उसको जैसे कि save करते हो वो auto restart server हो जाता है।

step (1)

Start the terminal

↓

npm install -g nodemon

↓

Then all done

Note:

जब आप localy काम करोगे तो इसको हर बार उस folder मे globaly install करना होगा। ताकि बार-बार server को Restart न करना पड़े।

Path Parameter

/:Name → To crate multiple route

→ देखो इसको समझने के लिए पुरा page पढ़ो।

जब हम instagram पे किसी page के लिए search करते है

↳ like

↳ https://www.instagram.com/apple

↳ https://www.instagram.com/microsoft

↳ https://www.instagram.com/username

इसका मतलब ऐसा नही है Rout path search के लिए developer ने लाखो और करोड़ो "get" response को send करना होगा। या लिखना होगा

```
app.get("/apple", (req, res) => {
    //code
});

app.get("/:username", (req, res) => {
    //code
});
```

देखो instagram पे billion user हैं तो ऐसा नहीं है कि billions 86 get response लिया गया होगा।

* Instagram Username search rout को as "Variable" treate करता है

with Database

index.js

```
Const express = require ("express");
Const app = express();
let port = 3002;
app.listen (port, () => {
    Console.log (`app liston a request ${port}`);
});
app.get ("/", (req, res ) => {
    res.send ("you connected with route path");
});

app.get ("/:Username/:id", (req, res) => {
    let {Username, id} = req.params;
    let htmlstr = '<h1> welcome ${Username}</h1>';
    res.send (htmlstr);
});
```

Common for all

PTO →

npm init → make new package

↓

npm install express → (listen req, give response)

↓

npm install -g nodemon → for (live server in local)

↓

save index.js

↓

nodemon index.js → for new run file

↓

server is live

↓

go to broser

↓

localhost:3002/AR/302 → search

↓

Find page

Dynamic routing like instagram "/:"

→ instagram मे जब आप जो search करते है उसके लिए ये नही है कि millons of code get response करता है वहाँ date base से befen करके आपको result show हो वही निचे sample of code है।

Previous code

index.js

* replace the previous get method of this and everything same

```
app.get("/:username", (req, res) => {
    const {username} = req.params;
    // Fetch user data from a database using the username
    //Example: const userdata = getUserData(username);
    if (userdata) {
        res.send(`<h1> welcome @${username}</h1>`);
    } else {
        res.status(404).send("User not found");
    }
});
```

Query route param

→ जब आप google या किसी npm के website पे आप कुछ भी search करते है तब वो एक query के तरह act करता है तो ऐसे response को करते using code ↓

like → https://www.google.com/search?=apple

```
app.get("/:search", (req, res) => {
    let {q} = req.query;
    if(!q) {
        return res.status(404).send("nothing exist");
    }
    res.send(`<h1> Here the about: ${q} </h1>`);
});
```

PTO

You have created own package.json

↓

You have install ~~the~~ express

↓

You have installed nodemon

↓

go to browser

↓

localhost:3002/search?=name

↓

we show the response

EJS

→ Embedded JavaScript templete ←

Templating → EJS ←

→ अब इसको resume templete कि help से समझते है। different-different user अपना details bill करते है resume बनकर ready हो जाता है उसे templete बोलते है।

→ Same बात है website मे जब same templete का काम हो तो उसके लिए multiple html code नही लिखते है। एक बार templete का code लिखते है।

→ अब instagram पर profile page सभी user के लिए same hi होता है follow button, following button, three dot button and many of layout same होता है for all user तो इसका मतलब यह नही है billions of user के billions of html page code लिखा गया होगा। इसी बड़े से काम को templeting करके एक blue print profile page बनाया जाता है और per user को उसके data के अनुसार merge करके show किया जाता है।

Note: यह सब संभव है using of EJS ये provide करता है to make template of blue print page layot।

Using EJS

process

make folder

↓

open in vs code

↓

npm init -y — default package.json file

↓

make index.js

↓

save all

↓

npm install express → download express npm package

↓

npm install nodemon → for live server to listen req

↓

npm i ejs → for templet the code

↓

All ready to code

↓

go index.js

index.js

```
Const express = require("express");
Const app = express();
let port = 3000;
~~app.listen(port,()=>{~~
app.listen(port,()=>{
    Console.log("server listening request");
});

app.set("view engine", "ejs");
app.get("/", (req, res)=>{
    res.render("home.ejs");
});
```

PTO break down code →

Note
जब हम ejs को use करते है तो express उसको bydefaut require कर लेता है

- EJS_Templete
  - node_modules
  - views
    - home.ejs
  - package-lock.json
  - package.json

```
app.set("view engine", "ejs");
```

↳ यह चाहता है कि जितने ejs templete बनेगे वो सभी एक views नाम के folder मे हो।

↳ folder का नाम views ही रखना ताकि ये express render करने के लिए आसानी से access कर सके।

```
app.get("/", (req, res) => {
    res.render("home.ejs");
});
```

↳ इसमे जब render करेगे home.ejs तो express सिधा views वाला folder मे जाकर ढुंढेगा क्योंकि express ejs file होने पर views नाम के folder मे से Access करता है और render करता है

Let me clear through diagram render template working

→ जब हम किसी website पे Home button पे click करते है तो direct home page पे redirect हो जाते है इसमे render template use होता है

Click the home button

↓

express waiting for request "home" → using is click button property

↓

express find home file in views folder → render the home file

↓

express found home in views express response → response show on the page

↓

home page

View directory

→ इसको include कर लेने से कभी भी view नाम को folder को Access करने मे error throw नही करती है, क्योंकि आप वह से भी Access कर सकते हैं और अन्दर से भी।

```
Const path = require ("path");
app.set ("views", path.join (__dirname, "/views"));
```

```
Const express = require ("express");
Const app = express();
Const path = require("path").
Const port = 8080;

app.set ("view engine", "ejs");
app.set ("view", path.join (__dirname, "\views"));

app.get ("/", (req, res) => {
    res.render ("home.ejs");
});

//code ---/

app.listen (port, () => {
    console.log ("listing server");
});
```

index.js

Interpolation Syntax

→ interpolation refers to embeding expressions into marked up text.

Eg – `This is ${name}`

→ value embeding

{add}

Note –

उदाहरण यहाँ पे ऐसे सा एक 'name' value embaded हो रहा है। लेकिन हम javaScript का full code को भी embed कर सकते हैं

Ejs tag

```
<body>
  <h1> <%= 2+2 %> </h1>
</body>
```

output

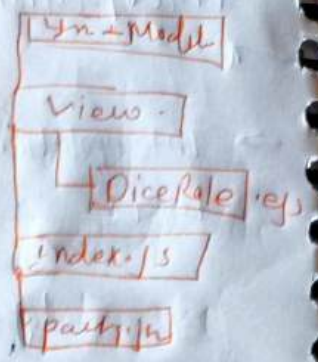
4

→ ये Ejs tag है जो कि उसके अन्दर लिखे हुए की value को javascript के तरह treat करता है और output को एक string return करता है

Passing data to ejs

database के data को store कर के render करना ejs file को views नाम के folder में

index.js




```
Const express = require("express");
  Const app = express();
Const path = require("path");
  let port = 3000;
app.listen (port, () => {
  Console.log("app is listing");
});

app.set ("view engine", "ejs");
app.set ("Views", path.join(__dirname, "/view"));
app.get("/", (req, res) => {
  let roldice = Math.floor(Math.random()*10)+1;
  res.render ("DiceRol.ejs", { roldice });
});
```

↳ view
↳ file

Suppose this data come from data base

Note 2

पिछले code मे जो file location show है वही same रहेगा। but jo home.ejs है उसके जगह पर views नाम के folder मे DiceVal.ejs रहेगा।

और है एक और बात जो math.floor का जो लिख कर हम कोई random Number को print कर रहे लेकिन suppose कि उसके जगह पर कोई 'data' database से आ रहा होगा। और DiceVal.ejs नाम के folder मे render हो रहा होगा।

bite

views
↳ DiceRole.ejs

```
<!DOCTYPE htm>
<html lang="en">
  <head>
    <meta ~ >
    <meta ~ >
    <title> Doumut </title>
  </head>
  <body>
    <h2> Dice Vale :- <% = dice %> </h2>
  </body>
</htm>
```

instagram EJS System

→ देखो जब हम हम लोग instagram पे किसी का profile search करते है तो billions of user है instagram का तो इसका ये मतलब नही है कि billions of profile के लिए billions of html code लिखना होगा।

→ with billion of code लिखे सम्भव है कि सभी का profile show किया जा सकता है using "EJS" render।

→ EJS database से data को collect कर के search किया गया username से match करने के बाद fetch करता है एक html blue print code से जिस एक html code से सभी user का profile show होता है।

Note

पिछले EJS templating code में लिखा गया folder path same होगा और code भी same है अभी हम सिर्फ इसका लिखेंगे।

http://localhost:3000/allu

Index.js

```
app.get("/:username",(req, res) => {
    let {username} = req.params;

    res.render("InstagramEJS.ejs", {username});
```

→ ये file view नाम के folder में होगा।

InstagramEJS.ejs

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta />
  </head>
  <title> ~ </title>
  <body>
      <h1> Hii dear @ <%= Username %> </h1>
      <button> follow </button>
      <button> Message </button>
  </body>
</html>
```

Conditional statement in EJS

EJS tag

https://ejs.co

→ we can write the JS code in ejs file of html.

→ Using some tags to perform operation, like { loop, if, else, array }

tag – "<%." → (this tag for control-flow, no output)

↳ loops, condition, if, else to control the condition and check it

Same hi file location rehega

index.js

```
app.get("/evenOrOdd", (req, res) => {
    let evenToOdd = Math.floor(Math.random() * 10) + 1;
    res.render("conditionStatement.ejs", {evenToOdd});
});
```

ConditionStatement.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta> ---
    <meta> --
<title> <title>
</head>
<body>
<h1> This is testing of condition in ejs </h1>

<% if(eventoOdd % 2 == 0) { %>
    <h1> Wow! even - <%= eventoOdd %> </h1>
<% }
    else { %>
    <h2> This is odd ! <%= eventoOdd %> </h2>
<% } %>
</body>
</html>
```

for loop in EJS

we edit and add something in instagram ejs file and index.js of the get method!

index.js

```
app.get("/:Username", (req, res) => {
    let {Username} = req.params;
  Const followers = ["allu", "ravi", "nitin"];
res.render("instagramEJS.ejs", {Username, followers});
});
```

View

→ instagramEJS.ejs

```
<body>
  <h2> followers </h2>
  <h3>
    <%. for (let name of followers) { %.>
      <UL>
        <Li> <%.= name %.> </Li>
      </UL>
    <%. } %.>
</h3>
```

VVI

Instagram page with data in EJS

EJS tag

<% Code js %> { control statement, loop }

<%= value %> { output value }

data.json – data base

instagram.ejs – Html page templete

index.js – Javasript file take req, on the base of request it give you response

index.js

```
app.get("/:username", (req, res) => {
  let {username} = req.Params;
  Const InstaData = require("./data.json");
  Const data = instaDate[username];
  if (data){
    res.render("instagram.EJS.ejs", {data});
  } else {
    res.status(404).send("User not found");
  }
});
```

data.json

```
}
"cats": {
    "name": "cats",
    "followers": 25000,
    "following": 5,
    "Posts": [
        {
            "image": "url",
            "likes": 200,
            "Comment": 17
        },
        {
            "image": "url",
            "likes": 2001",
            "Comment": 29
        },
        {
            "image": "url",
            "likes": 300,
            "Comment": 19
        }
    ]
},
"dogs": {
```

instagramEJS.ejs

```
<body>
    <h1>
        <%- if (data) { %>
        this is profile of @ <%= data.name %>
        <%} else { %>
            User not found
        <%} %>
    </h1>
<button> follow </button>
<button> followers </button>

<p> follow : <%= data.followers %>    
    followers : <%=data.following %> </p>
<hr>
<%- for (let pt of data.posts) { %>
    <img src = "" <%= pt.image %> alt="" ..
                style = "width :200px; height: 100px;">
    <p> like : <%= pt.likes %> pt.likes %>    
            Comment : <%= pt.Comments %> </p>
    <%} %>
</body>
```

Serving static files

```
app.use(express.static(folderName))
```

```
app.use(express.static(path.join(__dirname, "/public")))
```

ऐ लिखने से आप आप server को main folder से run कर सकते हो।

एक folder को Access करना हो तो

"/public"

दो से ज्यादा folder के अन्दर folder होते

"/public/style"

P10

Folder Arrangement

```
EJS_Templeate/
├→ node_modules/ (Node.js dependencies folder)
├→ public/ (Public folder for static asset)
│  ├→ CSS/
│  │  └→ style.CSS
│  └→ JS/
│     └→ app.JS
├→ views/ (EJS template file)
│  └→ instagram.ejs (EJS file for Instagram-related view)
├→ data.JSON (JSON file for data storge)
├→ index.JS (main server-side JavaScript file)
├→ package-lock.JSON (auto genrated file for locking dependency version)
└→ package.JSON (file manging dependencies)
```

index.js

```
app.Use (express.static (Path.join (__dirname, "/Public/Css")));
app.use(express.static(Path.join(__dirname, "/public/js")));
```

instagramEJS.ejs

```
<html>
<link rel="stylesheet" href="style.css" />
<body>
    //code
</body>
<script src="/app.js"></script>
</html>
```

Public/Css/style.sss

```
body {
background-color: yellow;
}
img {
    width: 200px;
    height: 200px;
}
```

Public/js/app.js

```
let btn = document.querySelectorAll
        ("button");
for (bt of btn) {
bt.addEventListener("click", () =>
    {
    console.log("btn was click")
    })
})
```

includes

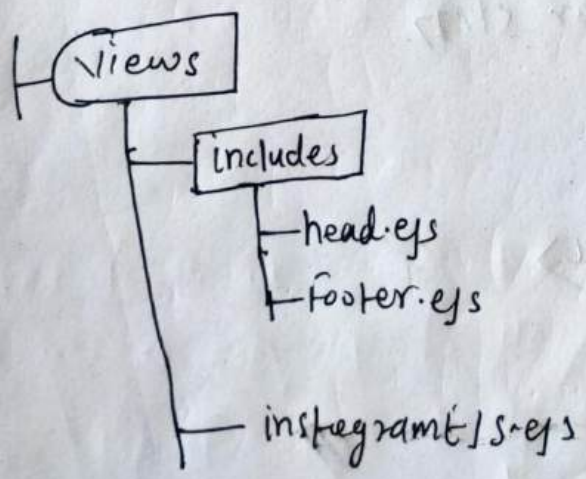
→ includes कि help से हमलोग कोई एक common code (some of code) जो कि बार-बार लिखना नही पड़े इसके लिए use. करते है।

Ex →

सभी html page मे header होते है जो सभी पेज के लिए same होते है तो उसको सभी files मे नही लिखकर उसे किसी folder मे ejs नाम से file save कर लेगे। और उसे header मे जो बाते है वो सभी मे लगाने के लिए।

```
<%- include("includes/head.ejs"); %>
```

folder ← file



instagramEls.ejs

```
<html>
<%- include("includes/head.ejs")%>
<body>
    // code
</body>
<%- include ("includes/footer.ejs") %>
</body>
<script src="/app.js"></script>
</html>
```

Footer.ejs

```
<div class="footer style="width:100%; height:100px;
                          background-color: blue;">
<h3> this part footer </h3>
</div>
```

Head.ejs

```
<!DOCTYPE html>
<---->
<head>
    <meta --
    <v ---
    <link rel="stylesheet" href="/style.css">
</head>
```

Miscellaneous

Get & post request

Get :-

Get used to get some response

Data send on query string

जब हम get का use करते है तो वे जो भी data है उसको URL मे show कर देता है।

इसका सही use है जब user के data को URL मे visible करना हो तब।

लेकिन user के द्वारा data गया Data को save करना हो तो get नही post use होता है।

Ex:-

```
<body>
    <form method="get" Action="/resgisfor">
      <input type="text" placeholder="username">
      <input type="password" placeholder="password">
      <button> Submit <button>
    </form>
</body>
```

जैसे कि submit करोगे username & password URL मे query के रूप मे add हो जाएगा।

Post :—

Used to post something (create/write/update)

Data send via request body (any type of data)

इसका use से कोई data को backend मे भेजा जाता है।

ये mostly Data send करता है via body request to the backend

```
<body>
    <form method="post" Action="resgistor">
    <input type="text" placeholder="user">
    <input type="password" placeholder="password">
    <button> Submit </button>
  </form>
</body>
```

जब submit होगा data तब data ना तो URL मे visible होगा और ना ही queystrin URL मे।

अब ये आप पे depend करता है कि हमे कहाँ use करना चाहते हो।

Previous Frontend code | 06 this is backend code

→ frontend में एक index.html होगा। और Backend में index.js और Backend folder में express, nodemon, npm को install कर लेना।

index.js

```
Const express = require("express");
Const app = express();

Const port = 3000;
// ये line post से भेजा गया data को catch करने में help करता है।
app.use (express.urlencoded ({extended : true}));
app.use (express.json());// User के द्वारा Json data को Accept करता है।

app.get ("/register", (req, res) => {

    Console.log (req.query);
    let {username, password} = req.query;
    res.send ('standard get response ${username}');
});
```

when we this must include urlencoded line for post

```
app.post ("/register", (req, res) => {
    Console.log (req.body);
    let {username, password} = req.body;
    res.send ('standard post request welcome ${username}');
});
app.listen (port, () => {
    Console.log ('port are listing ${port}');
});
```

<input type="text" name="Username" placeholder="User">

PTO → Understand better

Previous Understanding

Middleware ↓

```
app.use(express.urlencoded({extended: true}));
app.use(express.json());
```

Note ↓

देखो जब Middleware को add नही करोगे तो res.body data को read नही कर पाता है जिससे वो undefined show करता है

```
let {username, password} = req.body;
```

↳ ये data को तभी read करेगा जब upper आप middleware को use करेगे।