6.	Node	Templating - EJS interpolation syntax, data pass, static files, sub-templates - includes	41	ejs-mate (57)
7.	Node	GET & POST Requests	59	

6. Templating - EJS

→ EJS : Embedded JavaScript templates

→ EJS is a simple templating language that lets you generate HTML markup with plain Javascript

e.g. In Terminal

```
- HP@DP MINGW64 ~/Desktop / demo / EJSdir
  $ npm init -y  ← this flag used for setting a default values in package.json
  :                After created You can change its values if you want

- $ npm install express
  :

- $ npm install ejs
  :

- $ ~~#~~ touch index.js
- $ code index.js
- $ ls -la
  total ...
  ...   ...   ...  ./
  ...   ...   ...  ../
  ...   ...   ...  node_modules/
  ...   ...   ...  package-lock.json
  ...   ...   ...  package.json
  ...   ...   ...  index.js
```

In index.js

```
- const express = require("express");
  const app = express();

  const port = 8080 ;

  app.listen ( port,  () => {

       console.log(`App is listening on port ${port}`);
  });
```

In Terminal

```
- $ nodemon index.js
  :
  :
  App is listening on port 8080
```

Append in index.js

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

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

       res.render ("home.ejs");
  });
```

you can also write only "home" without ".ejs"

⇒ In EJSdir folder, make a new folder named "views"

⇒ By default, express require "ejs" automatically

⇒ By default, express searches for "views" named folder to run "render" function

⇒ We will make our .ejs file, ~~#~~ in this folder.

In demo/EJSdir/views/home.ejs

```
- <!DOCTYPE html>
  <html lang="en">
  :

   <body>
      <h1> This is home page</h1>
   </body>
  </html>
```

In Browser

- URL = localhost:8080

output will be as per "home.ejs" file

⇒ This is home page

* views directory

→ We can start server from parent directory, as below:

```
- HP@DP MINGW64 ~/Desktop/demo
$ nodemon EJSdir/index.js
```

→ Now, as per above command ~~e~~express cannot locate "views" folder, so we have to set path of "views" folder.

(Add) Append in index.js

```
- const path = require("path");

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

→ Now, it will work same as before.

* Interpolation Syntax

→ Interpolation refers to embedding expressions into marked up text.
(Just like string literals, we can write variables inside a string)

→ With the use of this, we can write js inside (inbetween) html syntax. (making ejs dynamic)

⇒ visit ejs.co website, scroll down to Tags section

<%	'Scriptlet' tag, for control-flow, no output
<%_	'Whitespace slurping' Scriptlet tag, strips all whitespace before it
<%=	Outputs the value into the template (HTML escaped) (output in string)
<%-	Outputs the unescaped value into the template
<%#	Comment tag, no execution, no output
<%%	Outputs a literal '<%'
%>	Plain ending tag
-%>	Trim-mode ('newline slurp') tag, trims following new line
_%>	'Whitespace slurping' ending tag, removes all whitespace after it

e.g.   add In views/home.ejs

```
- :
<body>
    <h1> This is home page </h1>

    <h3> <%= 1 + 2 %> </h3>
    <h3> <%= "apnacollege".toUpperCase() %> </h3>
    <h3> <%= ["one", "two"][1] %> </h3>

</body>
</html>
```

Output on :   This is home page
Web page   3
                APNACOLLEGE
                two

* Passing Data to EJS

eg. add In index.js

```
- app.get("/rolldice", (req, res) => {
    let num = Math.floor(Math.random() * 6) + 1 ;
    res.render("rollDice.ejs", { diceVal: num } );
});
```

No need to make two variable. see next page:

In EJSdir/views/rollDice.ejs

```
- <!DOCTYPE html>
  <html ... >
   :
   <body>
     <h1> Your Dice gave value: <%= diceVal %> </h1>
   </body>
  </html>
```

In BROWSER

- URL = localhost:8080/rolldice

Output: Your Dice gave value: 5
refresh
Your Dice gave value : 1

e.g. add in index.js

```
- app.get("/ig/:username", (req , res) => {

      let {username} = req.params;

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

In EJSdir/ views / instagram.ejs

```
- <!DOCTYPE html>
  < html ... >
      :
   <body>
       <h1> This page belongs to @ <%= username %></h1>
   </body>
  </html>
```

In Browser

- URL = localhost:8080/ig/parth

Output : This page belongs to @parth

* Conditional Statements in EJS

```
-      <%    if (diceVal == 6) {          %>.

              <h2> Nice! Roll Dice again </h2>

      <%    }     %>
```

→ In conditional statements, wrap it with tags of ejs without wrapping html markup

* Loops in EJS

- In index.js

```
- app.get("/ig/:username", (req, res) => {

        let {username} = req.params;

        const followers = ["name1", "name2", "name3"];

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

In EJSdir/views/instagram.ejs

```
- <h1> This page belongs to @<%= username %> </h1>
<h3> Accounts that follows you: </h3>
<ul> <%  for (let name of followers) {    %>
                <li> <%= name %> </li>
      <%    }    %>
</ul>
```

In BROWSER

- URL = localhost:8080/ig/italiya

Output : This page belongs to @italiya
Accounts that follows you:
· name1
· name2
· name3

* JSON Data in EJS

→ We have a json file which has some data and it is stored at EJSdir/data.json

→ e.g. A simple instagram page, we get data from json file and set it in .ejs file

In data.json

```
- {
    "cats": {
        "name": "cats",
        "followers": 25000,
        "following": 5,
        "posts": [
            {
              "image": "https://...",
              "likes": 200,
            }, "comments": 17
```

```
                {...},
                {...}
            ]
    },
    "dogs" : {
        "name" : "dogs",
        "followers" : 75000,
        "following" : 150,
        "posts" : [
            {...},
            {...},
            {...}
        ]
    }
}
```

In index.js

```
- app.get ( "/ig/:username" , (req,res) => {

      let {username} = req.params;

      const instaData = require ("./data.json");
      const data = instaData[username];

      if (data) {
            res.render ("instagram.ejs" , {data});
      } else {
            res.render("error.ejs");
      }
});
```

In views/error.ejs

```
:
:
<h1> No such Account </h1>
:
:
```

In views/instagram.ejs

```
:
:
<h1> This page belongs to @<%= data.name %> </h1>

<button> Follow </button>
<button> Message </button>

<p> Followers: <%= data.followers %> </p>
<p> Following: <%= data.following %> </p>

<hr>

<% for (let post of data.posts) {      %>

      <img src="<%= post.image %>" >
      <br>
      <p > Likes: <%= post.likes %>  &nbsp;&nbsp;&nbsp;
            Comments: <%= post.comments %>
      </p>

<% }  %>
:
```

In <u>Browser</u>

- URL = localhost:8080/ig/cats

Output: As per instagram.ejs page where username = cats

- URL = localhost:8080/ig/parth

Output: No such Account ← as per error.ejs

* Serving Static Files

→ In Best Practice, we will make a folder named "public" in "EJSdir" folder and store files of "css" and "js" respective folders.

In EJSdir/ public / js / app.js

```
- let buttons = document.querySelectorAll("button");

for ( const button of buttons ) {

      button.addEventListener("click",   () => {

            alert("Button was clicked");
      });
}
```

In EJSdir/ public /css / style.css

```
- body {
     background-color: aqua;
  }

img {
   height: 100px ;
   width: 100px ;
}
```

In EJSdir/index.js

```
- const express = require('express");
  const app = express ();
  const path = require("path");

  const port = 8080;

  app. listen ( ~~portt~~ port , () => {

        console.log(`App is listening on port ${port}`);
  });

→ app.use(express.static(path.join(__dirname,"/public/js")));
→ app.use(express.static(path.join(__dirname,"/public/css")));
```

OR If we starts server from EJSdir then ↓ ; If we start server from demo then ↑

```
→ app.use(express.static("/public/js"));
→ app.use(express.static("/public/css"));


  app.set("view engine", "ejs");
  app.set("views", path.join(__dirname, "/views"));
  :
  :
```

as per previous examples.... page : 51

In views/instagram.ejs $\boxed{IMP}$

- In head $\overset{element}{\text{~~tag~~}}$ ⇒ `<link rel="stylesheet" href="/style.css">`
  body $\overset{element}{\text{~~tag~~}}$ ⇒ as per page : 52 & at the this script tag just above end tag of body
  ⇒ `<script src="/app.js"></script>`

* Sub Templates (includes) (partials)

→ We have made home.ejs, error.ejs, instagram ... now, in these files we have written some code blocks same (similar).

→ These similar code blocks, we will remove from these files and store it in only only file, and we will call it sub templates.

→ In main .ejs files we will only write one line at appropriate place

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

e.g. In EJSdir/views/includes/footer.ejs

```
- <footer> This is footer </footer>
```

In EJSdir/views/includes/head.ejs.

```
-<head>
    <meta ... >
    <meta ... >
    <title> Home Page </title>
</head>
```

In EJSdir/ views / home.ejs

```
- <!DOCTYPE html>
  < html lang="en" >

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

  <body>
    <h1> This is home page </h1>

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

  </body >
  </html >
```

In EJSdir / views / error.ejs

```
-  ⋮
   <h1> No such Account </h1>

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

* ejs-mate package

- npm i ejs-mate

```
- const ejsMate = require("ejs-mate");
  app.engine("ejs", ejsMate);
```

P.T.O. →

→ boilerplate.ejs

```
– <html>
  <head> ... </head>
  <body>
     <h1> Something </h1>        ← instead of h1 you can include navbar.ejs
     <div class="container">
          <%-    body     %>     ← this is variable of all other ejs files
     </ div  >                   ← here, you can include footer.ejs
  </ body >
  </html>;
```

→ home.ejs

```
– <%  layout("boilerplate")   %>

<body>
     This is the body of home
</body>
```

→ new.ejs

```
– <%   layout("boilerplate")  %>

<body>
     This is the body of new
</body>
```

7. GET & POST Requests

→ GET : Used to get some response
 : Data sent in query strings, which is limited, is string data & is visible in URL.

→ POST : Used to post something (for create/write/update)
 : Data sent via request body which can be of any type.

e.g. 1   In demo/Misc/Frontend/ index.html

```
- <!DOCTYPE html >
< html Lang = "en" >
< head >
    < meta charset = " UTF-8 " >
    < meta name = "viewport"
           content = " width = device-width , initial-scale = 1.0 " >
    < title > GET & POST Request </ title >
</ head >
< body >

    < h3 > GET Request Form </ h3 >

    < form action = "/register" method = "get" >

     < input type = "text"
             placeholder = "enter username"
             name = "username" >
```

```
<input type="password"
       placeholder ="enter password"
       name ="password" >

<button type="submit"> GET Register </button>
</form>

<hr>

<h3> POST Request Form </h3>
<form action="/register method="post" >

   <input type="text"
          placeholder="enter username"
          name="username" >

   <input type="password"
          placeholder="enter password"
          name="password" >

   <button type="submit"> POST Register </button>

</form>

</body>
</html>
```

In BROWSER

- URL = 127.0.0.1:5500 ← (live server of vs code)
    - enter username = abc
    - enter password = 123
    - click on "GET Register" button

Output : URL = 127.0.0.1:5500/register?username=abc&password=123
on Webpage = Cannot GET /register

- URL = 127.0.0.1:5500
    - enter username = abcd
    - enter password = 1234
    - click on "POST Register" button

output : URL = 127.0.0.1:5500/register
on Webpage = This page isn't working right now.

↑ Default error page of browser

e.g.2 In Misc/Backend Terminal

```
- HP@DP MINGW64 ~/Desktop/Misc/Backend
  $ npm init -y
  :
  :
```

output: package.json file will be created in Backend folder. You can update value of the keys such as "author", ...

```
- $ npm install express

- $ touch index.js
- $ code index.js
```

In Misc/Backend/index.js

```
- const express = require("express");
  const app = express();

  const port = 8080;

  app.listen(port, () => {

      console.log{`App Is listening on port ${port}`);
  });
```

```
app. get ("/register" , (req , res) => {
     res. send ("This is GET response");
});

app. post ("/register", (req , res) => {
     res. send ("This is POST Response");
});
```

In Terminal

```
- $ nodemon index.js
  :
  App is listening on port 8080
```

In Browser (hoppscotch.io)

- HTTP verb = GET
  URL= http:// localhost : 2020 / register

  Output in : This is GET response
  Response body

- HTTP verb = POST
  URL = http:// localhost : 8080 / register

  Output in : This is POST response
  Response body

* Access data of GET Request

From a.q.1 @ c.1.2.

In Misc / Frontend / index.html

- Replace ⇒ action="/register"
  with ⇒ action = "http://localhost : 8080/ register"

In Misc / Backend / index.js

```
- app. get ("/register" , (req , res ) ⇒ {
      let { username , password } = req. query ;
      res. send (`This is GET Response, Welcome @ ${username}`);
});
```

In Browser

- URL = 127.0.0.1:5500
    - enter username = abc
    - enter password = 123
    - click on "GET Register" button

Output : URL = http:// localhost : 8080/register?username=abc&
password=123
on Webpage : This is GET response, Welcome @ abc

* ACCESS data of POST Request

from e.g 1 ① e.g. 2 ④ page. 64 e.g.

In Misc / Frontend / index.html

- We have already replaced value of action attribute

add In Misc / Backend / index.js

```
- app.use( express.urlencoded( {extended: true} ) );
app.post( "/register" , (req, res) => {
      :
    let {username , password } =  req.body ;

    res.send(`This is POST response, Welcome @${username}`);
});
```

In Browser

- URL = 127.0.0.1 : 5500
  - enter username = abcd
  - enter password = 1234
  - click on "POST Register" button

Output : URL = http:// localhost: 8080/ register

on webpage: This is POST response, Welcome @abcd

* Access JSON data of POST Request

From e.g. 1 ⊕ e.g. 2 ⊕ page 64 & 65 e.g.

add In Misc / Backend / index.js

```
- app.use( express.json() );
```

In Browser - (Hoppscotch.io)

- HTTP verb = POST

URL:= http:// localhost : 2020 / register

Body tab ⇒ Content Type = application / json

Raw Request Window = {

```
{
    "username": "parth",
    "password": "123"
}
```

Output in Response Body : This is POST response, Welcome @parth

* Data sent to POST as Object

```
- <form>
    <input name = "listing [username]" >
    <input name = "listing [password]" >
  </form>
```

```
- let newListing = req.body.listing;
          ↳ {
               username : "abcd",
               password : " 123abcd"
            }
```