

**NAME:**

**Roll No:**

**CS325 (JOEL02):: R-20 :: FULL STACK DEVELOPMENT**

NAME:  
Roll No:

## Lab 1:

Create a Node.JS environment with node and npm utilities commands and to check and test the node environment with Node.js Console module.

- **Step 1: installation of Node.js environment Node**

**1)Download Node.js:**

- 1)Visit the official Node.js website using your web browser.
- 2)On the homepage, you'll typically find download links for the latest version of Node.js. If you need a specific version, you might need to visit the "Previous releases" section.
- 3)Choose the appropriate installer for your operating system. Node.js provides installers for various platforms including Windows, macOS, and Linux.

**2)Install Node.js:**

- 1)Once the installer is downloaded, locate the downloaded file (usually in your "Downloads" folder).

Follow the installation instructions specific to your operating system:

**Windows:**

Double-click the downloaded installer file.

Follow the setup wizard instructions. You can generally accept the default settings, but ensure that the "npm package manager" option is selected during installation.

**3)Verify Installation:**

After installation, open a terminal or command prompt.

Type the following command to verify that Node.js and npm are installed correctly:

```
C:\Windows\System32\cmd.exe
D:\rvr>npm --version
11.2.0

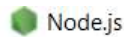
D:\rvr>node --version
v22.14.0

D:\rvr>|
```

**NAME:**  
**Roll No:**

These commands will display the versions of Node.js and npm installed on your system. If you see version numbers for both, the installation was successful.

- **Step 2: Test through the node REPL shell commands**
  - **Open Terminal/Command Prompt:** Open your terminal or command prompt.
  - **Check Node.js Installation:** Type `node -v` and press Enter to check if Node.js is installed correctly. It should display the version number.
  - **Open Node.js REPL:** Type `node` and press Enter to open the Node.js REPL (Read-Eval-Print Loop).
  - **Test Node.js Commands:** You can now test JavaScript commands directly in the REPL.



```
D:\rvr>node
Welcome to Node.js v22.14.0.
Type ".help" for more information.
> console.log('Hello, Welcome to FSD')
Hello, Welcome to FSD
```

- **Step-3: install prompt-sync module using npm utility.**
  - Install prompt-sync: In your terminal or command prompt, type:

```
D:\rvr>npm install prompt-sync

up to date, audited 4 packages in 11s

found 0 vulnerabilities
```

- **Step-4: Test and check the prompt-sync with console Module Application**

```
JS example_1.js X
JS example_1.js > ...
1  const prompt = require('prompt-sync')();|
2  const name = prompt('Enter the name:');
3  console.log(`Welcome, ${name}`);
```

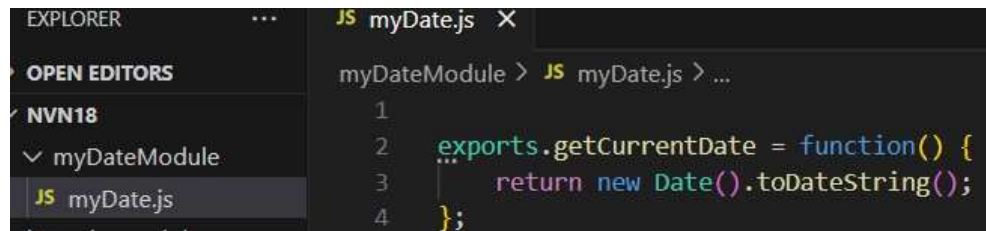
```
D:\rvr>node example_1.js
Enter the name:Ram
Welcome, Ram
```

NAME:  
Roll No:

## Lab 2:

Create a custom Date module using exports keyword Node module by using npm commands and to determine and display current Node.JS Webserver time and date.

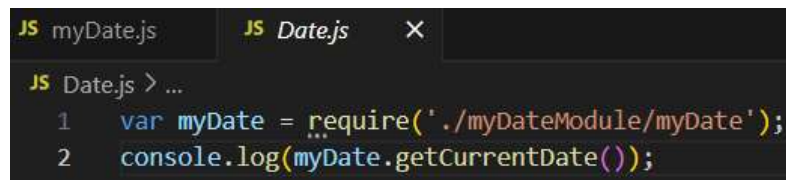
- **Step 1: Create Node Package Module myDate() using node utilities without package.json file**
  - 1) **Create a Directory:** Create a directory where you want to store your custom Node module. You can name it something like myDateModule.
  - 2) **Create myDate.js File:** Inside the directory, create a JavaScript file named myDate.js.
  - 3) **Define the Module:** In myDate.js, define your custom Date module.
  - 4) **Export the Module:** Use the exports keyword to make the getCurrentDate function accessible outside the module.



```
EXPLORER  ...  JS myDate.js X
OPEN EDITORS
NVN18
  myDateModule
    JS myDate.js
myDateModule > JS myDate.js > ...
1
2   exports.getCurrentDate = function() {
3       return new Date().toString();
4   };
```

```
D:\rvr>cd myDateModule
D:\rvr\myDateModule>node myDate.js
```

- 5) **Using the Module:** Now, you can use this module in other Node.js files by requiring it:



```
JS myDate.js  JS Date.js X
JS Date.js > ...
1   var myDate = require('./myDateModule/myDate');
2   console.log(myDate.getCurrentDate());
```

```
D:\rvr>node Date.js
Sun Mar 23 2025
```

NAME:  
Roll No:

- **Step -2 : Create the Node Package Module myDate() using with package.json file directives like version,name,bin,etc.,**

- Initialize a new npm project with npm init This will create ap package.json file.
- In the package.json file, you can specify directives like version, name, bin, etc

```
{
  "name": "mydatemodule",
  "version": "1.0.0",
  "description": "",
  "main": "myDate.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

- **Step – 3: Also install created packaged module using npm utility**

1)To install the module locally, you can use npm install <folder>. For example, if your module is in a folder named mydate, you would use npm install ./myDateModule.

```
D:\rvr>npm install ./myDateModule
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'super@0.1.0',
npm WARN EBADENGINE   required: { node: '>= 0.6.0 < 0.7.0' },
npm WARN EBADENGINE   current: { node: 'v20.11.0', npm: '10.2.4' }
npm WARN EBADENGINE }

added 1 package, and audited 8 packages in 2s

found 0 vulnerabilities
```

2)To use the module in your code, you can now use require('myDateModule').

```
JS Date.js  X
JS Date.js > ...
1  var myDate = require('myDateModule');
2  console.log(myDate.getCurrentDate());|
```

```
D:\rvr>node Date.js
Sun Mar 23 2025
```

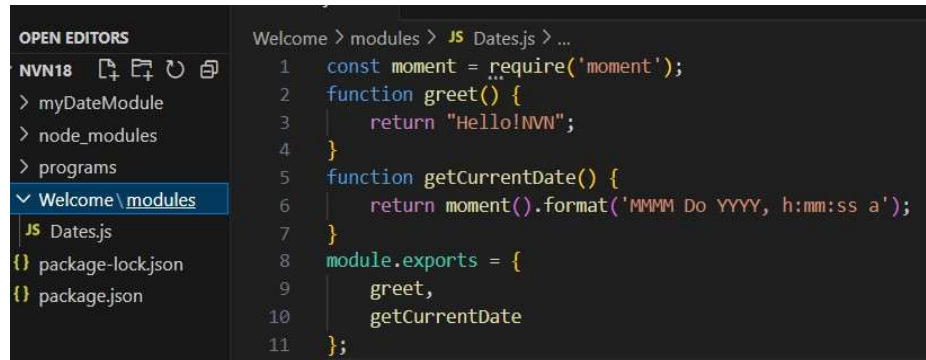
NAME:  
Roll No:

### Lab 3:

Create Node JS Application with Folder structure using npm utilities and develop one application to display “welcome Node JS APP” Greet message

- Step -1 : With VisualStudioCode APP Framework(Any other)

- 1) Create a Folder named Welcome and Create the another Folder inside the Welcome named Modules , inside the Modules create the file named Dates.js in Vs Code.



```
OPEN EDITORS
NVN18 [🔍] [🔍] [🔄] [🗑️]
> myDateModule
> node_modules
> programs
▼ Welcome\modules
  JS Dates.js
  {} package-lock.json
  {} package.json

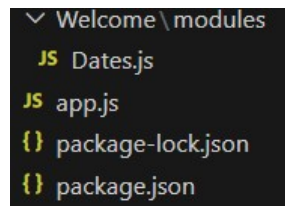
Welcome > modules > JS Dates.js > ...
1  const moment = require('moment');
2  function greet() {
3      return "Hello!NVN";
4  }
5  function getCurrentDate() {
6      return moment().format('MMMM Do YYYY, h:mm:ss a');
7  }
8  module.exports = {
9      greet,
10     getCurrentDate
11 };

```

```
D:\rvr\Welcome\modules>node Dates.js

D:\rvr\Welcome\modules>
```

- 2) Create the app.js file at Outside the Welcome Folder and run the program using the node .



```
▼ Welcome\modules
  JS Dates.js
  JS app.js
  {} package-lock.json
  {} package.json

```



```
JS app.js X
JS app.js > ...
1  const moment = require('moment');
2  const myDate = require('./Welcome/modules/Dates');
3
4  console.log("Welcome to Node.js APP");
5  console.log("Greet Message:", myDate.greet());
6  console.log("Current Date:", myDate.getCurrentDate());
7

```

```
D:\rvr\Welcome\modules>cd ..

D:\rvr\Welcome>cd ..

D:\rvr>node app.js
Welcome to Node.js APP
Greet Message: Hello!NVN
Current Date: April 11th 2024, 9:09:07 pm

```

NAME:  
Roll No:

- **Step – 2: Without VisualStudioCode APP Framework**

1) Create a Directory named Welcome and Create the another Directory inside the Welcome named Modules , inside the Modules create the file named Dates.js without Vs Code.

```
D:\rvr>mkdir Welcome
D:\rvr>cd Welcome
D:\rvr\Welcome>mkdir modules
D:\rvr\Welcome>cd modules
D:\rvr\Welcome\modules>
```

```
const moment = require('moment');
function greet() {
    return "Hello!NVN, Let's Code";
}
function getCurrentDate() {
    return moment().format('MMMM Do YYYY, h:mm:ss a');
}
module.exports = {
    greet,
    getCurrentDate
};
```

```
D:\rvr\Welcome\modules>node Dates.js
D:\rvr\Welcome\modules>
```

2) Outside the Welcome Directory Create the app.js File to run the main program:

```
JS app.js
JS app.js > ...
1  const moment = require('moment');
2  const myDate = require('./Welcome/modules/Dates');
3
4  console.log("Welcome to Node.js APP");
5  console.log("Greet Message:", myDate.greet());
6  console.log("Current Date:", myDate.getCurrentDate());
7
```

C:\Windows\System32\cmd.exe

```
D:\rvr\Welcome\modules>cd ..
D:\rvr\Welcome>cd ..
D:\rvr>node app.js
Welcome to Node.js APP
Greet Message: Hello!NVN, Let's Code
Current Date: April 11th 2024, 9:35:07 pm
```

NAME:

Roll No:

- **Step – 3: Also Access the Custom myDate Module.**

- 1) Create a file named myDates.js , where this code exports a function myDates.js that returns current date and time.

```
JS myDates.js X JS apps.js
JS myDates.js > myDates > myDates
1 exports.myDates = function()
2 {
3     return Date();
4 }
```

```
D:\rvr>node myDates.js
```

```
D:\rvr>|
```

- 2)import the myDates.js using the require in the apps.js file :

```
JS apps.js > ...
1 var dt = require('./myDates.js');
2
3 console.log('welcome to NODEJS');
4 console.log('Current time and Date is:'+dt.myDates());|
```

```
D:\rvr>node apps.js
welcome to NODEJS
Current time and Date is:Thu Apr 11 2024 21:55:11 GMT+0530 (India Standard Time)
```



NAME:  
Roll No:

## Lab 4:

Create Angular CLI Applications with different component configuration steps using different @Angular ng module utilities at CLI environment.

- Step -1 :Class component Angular app

- 1) First install the angularjs using the command prompt as show in the below:

```
C:\WINDOWS\system32\cmd.exe  
D:\rvr>npm install -g @angular/cli  
added 273 packages in 2m  
52 packages are looking for funding  
run 'npm fund' for details
```

- 2) Create a new Angular Application named myApp with the following syntax:

```
D:\rvr> ng new myApp
```

- 3) Create a new Component with name of the myComponent:

```
D:\rvr> cd myApp  
D:\rvr\myApp>ng generate component myComponent
```

- Step – 2: Define Inline selector component in Angular HelloWorld app with root element

- 1) Open my-component.component.ts in your Angular application and by default the selector value will be 'app-component' but change the selector property to 'app-root'. Your my-component.component.ts file should look like this:

```
App > src > app > my-component > TS my-component.component.ts > MyComponent  
1  import { Component } from '@angular/core';  
2  
3  @Component({  
4    selector: 'app-root',  
5    standalone: true,  
6    imports: [],  
7    templateUrl: './my-component.component.html',  
8    styleUrls: ['./my-component.component.css']  
9  })  
10 export class MyComponentComponent {  
11  
12  }  
13
```

NAME:

Roll No:

- **Step – 3: Define Inline template component in Angular HelloWorld app with HTML elements**

- 1) In my-component.component.ts, change the templateUrl property to template and define your HTML elements inline. Your my-component.component.ts file should look like this:

```
myApp > src > app > my-component > TS my-component.component.ts > MyComponentComponent
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    standalone: true,
6    imports: [],
7    template: `
8      <h1>HelloWorld! Welcome to Angular</h1>
9      <p>This is an inline template.</p>
10   `,
11    styleUrls: ['./my-component.component.css']
12  })
13  export class MyComponentComponent {}
14
15
```

- **Step – 4: Define Inline Style component in Angular HelloWorld app to style the color of the message.**

- 1) In my-component.component.ts, change the StyleUrl property to styles and define your CSS elements inline. Your my-component.component.ts file should look like this :

```
myApp > src > app > my-component > TS my-component.component.ts > M
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    standalone: true,
6    imports: [],
7    template: `
8      <h1>HelloWorld! Welcome to Angular</h1>
9      <p>This is an inline template.</p>
10     <p class="message">This is an inline template.</p>
11   `,
12    styles: [
13      `
14      .message {
15        color: red;
16      }
17   `
18  ])
19  export class MyComponentComponent {
20  }
21
```

**NAME:**

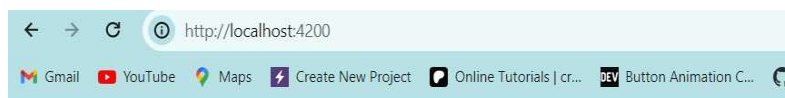
**Roll No:**

2) Run the application using the following command:

```
D:\rvr\myApp>ng serve
Initial chunk files | Names          | Raw size
polyfills.js       | polyfills      | 83.60 kB |
main.js            | main           | 1.81 kB  |
styles.css         | styles         | 95 bytes |
                   | Initial total  | 85.50 kB

Application bundle generation complete. [8.864 seconds]

Watch mode enabled. Watching for file changes...
  Local:  http://localhost:4200/
  press h + enter to show help
```



# HelloWorld! Welcome to Angular

HTML TEMPLATE - This is an inline template.

STYLE TEMPLATE - This is an inline template.

NAME:  
Roll No:

## Lab 5:

Create Angular CLI Applications using Angular Class component constructors and objects and different variable initialization.

- Step – 1: Create Date Class Constructor with current Date in Class Component

1) Create a new Angular project: you can create a new Angular project by running:

```
D:\rvr>ng new Lab --no-standalone
? Which stylesheet format would you like to use? CSS
? Do you want to enable Server-Side Rendering (SSR) and Static
```

2) Create a new component: Navigate into your new project directory and generate a new component. For example, if we want to create a component named date, we would run:

```
D:\rvr>cd Lab

D:\rvr\Lab>ng g c date
CREATE src/app/date/date.component.html (20 bytes)
CREATE src/app/date/date.component.spec.ts (610 bytes)
CREATE src/app/date/date.component.ts (201 bytes)
CREATE src/app/date/date.component.css (0 bytes)
UPDATE src/app/app.module.ts (543 bytes)
```

3) Update the Component Class: Open the date.component.ts file and update it as follows:

```
Lab > src > app > date > TS date.component.ts > ...
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-date',
5    templateUrl: './date.component.html',
6    styleUrls: ['./date.component.css']
7  })
8  export class DateComponent implements OnInit {
9    currentDate: Date;
10
11    constructor() {
12      this.currentDate = new Date();
13    }
14
15    ngOnInit(): void {
16    }
17  }
```

After the updation of this file , print the Current Date in the date.component.html file as follows:

NAME:

Roll No:

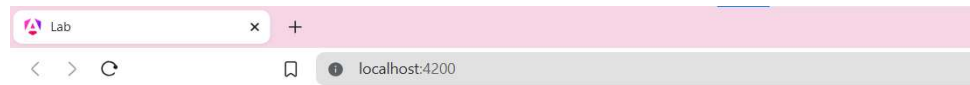
```
Go to component
1 <h1>Mr.NVN, The date works!</h1>
2 <h2>the current Date is {{currentDate}}</h2>
```

At this point , the newly generated component date is completed , to run this application , add the <app-date> selector in the app.component.html

```
TS app.module.ts app.component.html X
Lab > src > app > app.component.html > ...
Go to component
1
2 <app-date></app-date>
```

**4)Run the Application:** To run the application , use the following command:

```
D:\rvr\Lab>ng serve
Browser bundles
Initial chunk files | Names | Raw size
polyfills.js | polyfills | 83.60 kB |
main.js | main | 3.62 kB |
styles.css | styles | 95 bytes |
| Initial total | 87.31 kB
```



**Mr. RAM , The date works!**

the current Date is Sun Mar 23 2025 13:51:13 GMT+0530 (India Standard Time)

- **Step – 2: By using Selector, templateUrl and styleUrls External component configurations demonstrate the constructor with different objects**

**1)Create a new component:** Navigate into your new project directory and generate a new component. Here we created the userDetails component:

```
D:\rvr\Lab>ng g c users
CREATE src/app/users/users.component.html (21 bytes)
CREATE src/app/users/users.component.spec.ts (617 bytes)
CREATE src/app/users/users.component.ts (205 bytes)
CREATE src/app/users/users.component.css (0 bytes)
UPDATE src/app/app.module.ts (626 bytes)
```

NAME:

Roll No:

2) Update the component class: update the users.component.ts file, like this:

```
ab > src > app > users > TS users.component.ts > UsersComponent > cons
1  import { Component,OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-users',
5    templateUrl: './users.component.html',
6    styleUrls: ['./users.component.css']
7  })
8  export class UsersComponent implements OnInit {
9    user: {name: string, age: number, class: string};
10
11    constructor() {
12      this.user = {
13        name: 'NVN',
14        age: 18,
15        class: 'CSE-C'
16      };
17    }
18
19    ngOnInit(): void {
20    }
21  }
```

3) Create the html file: update the users.component.html file in the users folder.

```
ab > src > app > users > users.component.html > html > body > p
Go to component
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>Document</title>
7  </head>
8  <body>
9    <h1>Users Details</h1>
10   <p>UserName:{{user.name}}</p>
11   <p>Age:{{user.age}}</p>
12   <p>Class:{{user.class}}</p>
13 </body>
14 </html>
15
```

4) Run the Application: Now, To Run the Application , use the following command:





NAME:  
Roll No:

## Lab 6:

Create Angular CLI Applications using Angular Expressions and Filters to demonstrate the one App.

- Create different Angular Expressions in Class Component
- Also Specify with Different Angular pipes or filters to demonstrate each filter with Angular expression

1) Create the angular application with name of Expr\_Filter , with the following command :

```
D:\rvr>ng new Expr_Filter --no-standalone
? Which stylesheet format would you like to use? CSS
? Do you want to enable Server-Side Rendering (SSR) and
```

2) Create the new component for the Expr\_Filter Application with name of Expression , with the following command :

```
D:\rvr>cd Expr_Filter

D:\rvr\Expr_Filter>ng generate component Expression
CREATE src/app/expression/expression.component.html (26 bytes)
CREATE src/app/expression/expression.component.spec.ts (652 bytes)
CREATE src/app/expression/expression.component.ts (225 bytes)
CREATE src/app/expression/expression.component.css (0 bytes)
UPDATE src/app/app.module.ts (567 bytes)
```

3) Write the expression and filters code in the expression.component.ts and bind those values in the expression.component.html files , the code follows as:

```
Expr_Filter > src > app > expression > TS expression.component.ts > ExpressionComponent
1  import { Component ,OnInit} from '@angular/core';
2
3  @Component({
4    selector: 'app-expression',
5    templateUrl: './expression.component.html',
6    styleUrls: ['./expression.component.css']
7  })
8  export class ExpressionComponent implements OnInit {
9    user: {name: string, age: number,class: string};
10    title:string;
11    currency:number;
12    date:Date;
13    constructor() {
14      this.user = {
15        name: 'NVN',
16        age: 18,
17        class: 'CSE-C'
18      };
19      this.title='Expression and Filters';
20      this.currency = 106106;
21      this.date = new Date();
22    }
23    ngOnInit(): void {
24      throw new Error('Method not implemented.');
```

NAME:

Roll No:

3) To execute the pipes, write the following code in the expression.component.html in the following way:

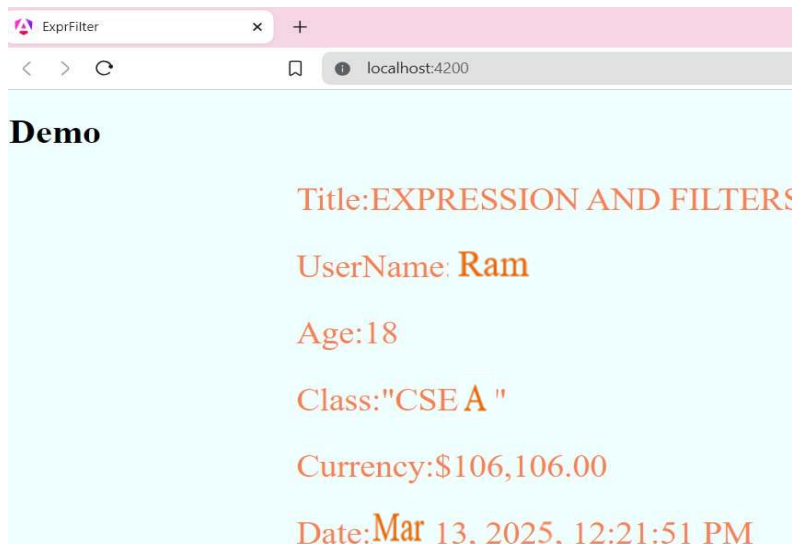
```
Go to component
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <p>Title:{{title|uppercase}}</p>
  <p>UserName:{{user.name | lowercase}}</p>
  <p>Age:{{user.age | number}}</p>
  <p>Class:{{user.class | json}}</p>
  <p>Currency:{{currency|currency}}</p>
  <p>Date:{{date|date:"medium"}}</p>
</body>
</html>
```

4) Before that link the expression.component.html file to app.component.html main file, in the following way:

```
<> app.component.html • TS expression.compone
Expr_Filter > src > app > <> app.component.html > ...
Go to component
1 |
2 <app-expression></app-expression>
```

5) Execute the angular application, with the following command :

```
D:\rvr>cd Expr_Filter
D:\rvr\Expr_Filter>ng serve
Browser bundles
Initial chunk files
```





NAME:  
Roll No:

## Lab 7:

Create Angular CLI Applications using Data Binding demonstrate each binding type with form elements.

- Interpolation Binding.
- Style Binding
- Class Binding.
- Two –way binding.

1) Create the angular application with name binding , in the following way:

```
D:\rvr>ng new binding --no-standalone
? Which stylesheet format would you like to use? CSS
? Do you want to enable Server-Side Rendering (SSR) and Stat
```

2) In the app.component.ts file make the following changes to do the binding process:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = `Interpolation||
           Style Binding||
           Class Binding||
           Two-Way Binding`;
  color='blue';
  isSpecial: boolean = true;
  inputValue='';
}
```

3) All the binding modules will work , except the Two-Way binding , so make the changes in the app.module.ts file.

In the app.module.ts file , import the FormsModule in that file , in the following way:

```
import { NgModule } from '@angular/core';
import { BrowserModule, provideClientHydration } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [
    provideClientHydration()
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

NAME:

Roll No:

- 4) In the app.component.html , write the following code to binding the elements into webpage:

```
app.component.html X # app.component.css TS app.component.ts TS app.module.ts
inding > src > app > < app.component.html > html > body
Go to component
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9   <div id="interpolation">
10    <!--Interpolation Binding-->
11    <h2>This is the Interpolation</h2>
12    <h1>{{title}}</h1>
13  </div>
14
15  <div id="style">
16    <!--Style Binding-->
17    <h3>This is the Style Binding</h3>
18    <p [style.color]="color">This the style binding</p>
19  </div>
20
21  <div id="class">
22    <!--Class Binding-->
23    <h3>This is the Class Binding</h3>
24    <p [class.special]="isSpecial">This is a paragraph with the Class Binding</p>
25  </div>
26  <div id="two-way">
27    <!--Two-Way Binding-->
28    <h3>This is the Two-Way Binding</h3>
29    <input [(ngModel)]="inputValue" placeholder="Enter text">
30    <p>Your input: {{ inputValue }}</p>
31  </div>
32 </body>
33 </html>
```

- 5) To Run the Application do the following :

```
D:\rvr>cd binding
D:\rvr\binding>ng serve
Browser bundles
Initial chunk files | Names
polyfills.js       | polyfills
main.js            | main
```

NAME:

Roll No:

Binding

localhost:4200

**This is the Interpolation**

**Interpolation|| Style Binding|| Class Binding|| Two-Way Binding**

**This is the Style Binding**

This the style binding

**This is the Class Binding**

This is a paragraph with the Class Binding

**This is the Two-Way Binding**

Your input: NVN

NAME:  
Roll No:

## Lab 8:

Create Node.js Application using URL module to decompose URL Components with  
`urlStr =`

`'http://user:pass@host.com:80/resource/path?query=string#ha'`

- Resolving the URL Components with `url.parse()` and `url.format()` methods
- Also to Resolving the URL using `url.resolve()`;

- 1) **Create a New Node.js Project:** Open your terminal and create a new directory for your project. Then navigate into that directory and initialize a new Node.js project by running:

```
D:\>cd nvvn18
D:\rvr>mkdir URL
D:\rvr>cd URL
D:\rvr\URL>npm init -y
```

- 2) **Install Required Dependencies:** Since we'll be using built-in Node.js modules, there are no external dependencies to install.

```
D:\rvr\URL>npm install url
added 17 packages, and audited 18 packages in 4s
11 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
```

- 3) **Create the Node.js Script:** Create a new JavaScript file, named **url.js** such as `app.js`, in your project directory. This file will contain the code to decompose and resolve URLs.

Write the code in the following ways:

NAME:  
Roll No:

```
const url = require('url');
const urlStr = 'http://user:pass@host.com:80/resource/path?query=string#ha';
const parsedUrl = url.parse(urlStr);

console.log('Decomposed URL Components:');
console.log('Protocol:', parsedUrl.protocol);
console.log('Username:', parsedUrl.auth.split(':')[0]);
console.log('Password:', parsedUrl.auth.split(':')[1]);
console.log('Host:', parsedUrl.host);
console.log('Port:', parsedUrl.port);
console.log('Path:', parsedUrl.pathname);
console.log('Query:', parsedUrl.query);
console.log('Hash:', parsedUrl.hash);
console.log('-----');
const resolvedUrl = url.format(parsedUrl);
console.log('Resolved URL:');
console.log(resolvedUrl);
console.log('-----');
const resolvedPath = url.resolve('http://example.com/', '/resource');
console.log('Resolved Path:');
console.log(resolvedPath);
```

4) Run the Application: Run the url.js application , in the following way:

```
D:\rvr\URL>node url.js
Decomposed URL Components:
Protocol: http:
Username: user
Password: pass
Host: host.com:80
Port: 80
Path: /resource/path
Query: query=string
Hash: #ha
-----
Resolved URL:
http://user:pass@host.com:80/resource/path?query=string#ha
-----
Resolved Path:
http://example.com/resource
```

**NAME:**  
**Roll No:**

## Lab 9:

Implementing Http Server and Http Client using http node.js module and demonstrate the Http Client/server Application.

- Create Http Static server files data using static files.
- Define HttpRequest/HttpResponse objects.
  - 1) **Create the necessary files:** Create a new directory for your project and create the http\_server.js and http\_client.js files. Also, create a public directory and place server.html file in it. You can put some basic HTML content in server.html.

```
D:\rvr>mkdir HTTP
```

```
D:\rvr>cd HTTP
```

```
D:\rvr\HTTP>mkdir public
```

```
D:\rvr\HTTP>cd public
```

```
D:\rvr\HTTP\public>cd ..
```

- 2) Create the http\_server.js files and http\_client.js files and write the following code Mentioned below and at the same time create the server.html
- 3) **Run the server:** Open a terminal, navigate to your project directory, and run node http\_server.js. You should see the message 'Server is listening on port 3000'.
- 4) **Test the server:** Open a web browser and go to http://localhost:3000. You should see the content of your index.html file.
- 5) **Run the client:** In a new terminal window (or tab), navigate to your project directory and run node http\_client.js. You should see the content of your server.html file printed in the terminal. This is the response from the server.



NAME:  
Roll No:

http\_server.js:

```
JS url.js JS http_client.js JS http_server.js ● server.html
HTTP > JS http_server.js > ...
1  const http = require('http');
2  const fs = require('fs');
3  const path = require('path');
4
5  const server = http.createServer((req, res) => {
6      // Serve static files
7      const filePath = path.join(__dirname, 'public', req.url === '/' ? 'server.html' : req.url);
8      const contentType = getContentType(filePath);
9      // Check if file exists
10     fs.readFile(filePath, (err, content) => {
11         if (err) {
12             if (err.code === 'ENOENT') {
13                 res.writeHead(404, { 'Content-Type': 'text/html' });
14                 res.end('<h1>404 Not Found</h1>');
15             } else {
16                 res.writeHead(500); (parameter) err: NodeJS.ErrnoException
17                 res.end(`Server Error: ${err.code}`);
18             }
19         } else {
20             res.writeHead(200, { 'Content-Type': contentType });
21             res.end(content, 'utf8');
22         }
23     });
24 });
25 const PORT = process.env.PORT || 3000;
26
27 server.listen(PORT, () => console.log(`Server running on port ${PORT}`));
28 // Function to determine content type based on file extension
29 function getContentType(filePath) {
30     let extname = path.extname(filePath);
31     switch (extname) {
32         case '.html':
33             return 'text/html';
34         case '.js':
35             return 'text/javascript';
36         case '.css':
37             return 'text/css';
38         case '.json':
39             return 'application/json';
40         case '.png':
41             return 'image/png';
42         case '.jpg':
43             return 'image/jpg';
44         default:
45             return 'text/plain';
46     }
47 }
```

NAME:

Roll No:

http\_client.js:

```
url.js JS http_client.js X JS http_server.js <> s
HTTP > JS http_client.js > ...
1  const http = require('http');
2  const options = {
3    hostname: 'localhost',
4    port: 3000,
5    path: '/',
6    method: 'GET'
7  };
8  const req = http.request(options, (res) => {
9    let data = '';
10   res.on('data', (chunk) => {
11     data += chunk;
12   });
13   res.on('end', () => {
14     console.log('Response:', data);
15   });
16 });
17 req.on('error', (error) => {
18   console.error('Error:', error);
19 });
20 req.end();
```

Server.html:

```
HTTP > public > <> server.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>Document</title>
7  </head>
8  <body>
9    <p>This is the port running</p>
10 </body>
11 </html>
```



NAME:  
Roll No:

```
C:\WINDOWS\system32\cmd.exe - node http_server.js  
  
D:\rvr\HTTP>node http_server.js  
Server running on port 3000
```



THis is the port running

```
C:\> Select C:\WINDOWS\system32\cmd.exe  
  
D:\rvr\HTTP>node http_client.js  
Response: <!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Document</title>  
</head>  
<body>  
  <p>THis is the port running</p>  
</body>  
</html>
```

NAME:  
Roll No:

## Lab 10.

Create Simple Arithmetic Operations Form with different form input elements N1 and N2 text components and ADD button component.

- provide Express Server with listen port:3000
- Use Express.use route and URL Pattern '/add'
- provide different routing configurations either POST or GET

- 1) Create the directory ADD and inside the directory create the node application with name of add.js .

```
D:\rvr>mkdir ADD
```

```
D:\rvr>cd ADD
```

```
D:\rvr>npm install express
added 64 packages, and audited 67 packages in 5s
12 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
```

- 2) We import Express and the body-parser middleware for parsing form data.
- 3) We create an instance of the Express application.
- 4) We set the port to 3000.
- 5) We use bodyParser.urlencoded() middleware to parse URL-encoded form data.
- 6) We define a route for serving the HTML form (/), which contains two input fields for numbers (N1 and N2) and a submit button (ADD).
- 7) We define a route for handling POST requests to /add. When the form is submitted, this route extracts the numbers from the form data, adds them together, and sends the result as a response.
- 8) We optionally define a route for handling GET requests to /add. This route is similar to the POST route but expects the numbers to be passed as query parameters instead of form data.
- 9) We start the server and listen on port 3000.
- 10) The code will be followed as :

NAME:  
Roll No:

```
ADD > JS add.js > ...
1  const express = require('express');
2  const bodyParser = require('body-parser');
3  const app = express();
4  const PORT = 3200;
5  app.use(bodyParser.urlencoded({ extended: true }));
6  app.get('/', (req, res) => {
7      res.send(`
8          <form action="/add" method="POST">
9              <label for="n1">Enter number 1:</label>
10             <input type="text" id="n1" name="n1"><br>
11             <label for="n2">Enter number 2:</label>
12             <input type="text" id="n2" name="n2"><br>
13             <button type="submit">ADD</button>
14         </form>
15     `);
16 });
17 app.post('/add', (req, res) => {
18     const n1 = parseFloat(req.body.n1);
19     const n2 = parseFloat(req.body.n2);
20     const result = n1 + n2;
21     res.send(`The result of adding ${n1} and ${n2} is ${result}`);
22 });
23 app.listen(PORT, () => {
24     console.log(`Server is running on http://localhost:${PORT}`);
25 });
```

```
D:\rvr\ADD>node add.js
Server is running on http://localhost:3200
```

The image shows two screenshots of a web browser. The top screenshot shows the initial form at localhost:3200. It has two text input fields labeled 'Enter number 1:' and 'Enter number 2:'. The first field contains the number '20' and the second field contains the number '30'. Below these fields is a button labeled 'ADD'. The bottom screenshot shows the result page at localhost:3200/add. It displays the text 'The result of adding 20 and 30 is 50'.

The result of adding 20 and 30 is 50

NAME:  
Roll No:

## Lab 11:

Create Simple Login form Page Application using Express JS Module:

- provide Express Server with listen port:4000 with URL Pattern '/login'
- Display the login form with username, password, and submit button on the screen.
- Users can input the values on the form.
- Validate the username and password entered by the user.
- Display Invalid Login Credentials message when the login fails.
- Show a success message when login is successful.

- 1) Create the directory named LOGIN and inside the directory create the file named login.js

```
D:\rvr>mkdir LOGIN
```

```
D:\rvr>cd LOGIN
```

```
D:\rvr>npm install express
added 64 packages, and audited 67 packages in 5s
12 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
```

- 2) We import Express and the body-parser middleware for parsing form data.
- 3) We create an instance of the Express application.
- 4) We set the port to 4000.
- 5) We use bodyParser.urlencoded() middleware to parse URL-encoded form data.
- 6) We define a route for serving the login form (/login). This route displays a form with input fields for username and password, along with a submit button.
- 7) We define a route for handling POST requests to /login. This route receives the submitted form data, validates the username and password, and sends an appropriate response:
- 8) If both username and password are provided and match the expected values (in this case, 'admin' and 'password'), it sends a success message.
- 9) If either username or password is missing, it sends a message prompting the user to enter both.
- 10) If the provided username or password is incorrect, it sends an error message indicating invalid credentials.
- 11) We start the server and listen on port 4000.
- 12) The code follows as :

NAME:  
Roll No:

Login.js:

```
LOGIN > .\login.js > ...
1  const express = require('express');
2  const bodyParser = require('body-parser');
3  const app = express();
4  const PORT = 4000;
5  app.use(bodyParser.urlencoded({ extended: true }));
6  app.get('/login', (req, res) => {
7      res.send(`
8          <h1>Login</h1>
9          <form action="/login" method="POST">
10             <label for="username">Username:</label>
11             <input type="text" id="username" name="username"><br>
12             <label for="password">Password:</label>
13             <input type="password" id="password" name="password"><br>
14             <button type="submit">Login</button>
15         </form>
16     `);
17 });
18 app.post('/login', (req, res) => {
19     const { username, password } = req.body;
20     if (username && password) {
21         if (username === 'nvn' && password === 'emma') {
22             res.send('<h1>Login Successful!</h1>');
23         } else {
24             res.send('<h1>Invalid Login Credentials</h1>');
25         }
26     } else {
27         res.send('<h1>Please enter both username and password</h1>');
28     }
29 });
30 app.listen(PORT, () => {
31     console.log(`Server is running on http://localhost:${PORT}`);
32 });
```

```
D:\rvr\LOGIN>node login.js
Server is running on http://localhost:4000
```

NAME:  
Roll No:



# Login

Username:

Password:

Login



# Login Successful!



NAME:  
Roll No:

## Lab 12:

**Create Simple MongoDB Server with mongod configuration data and also manage Mongoshell using mongosh :**

- Create simple student document Database
- Insert one student record in mongosh
- Update and delete one document in mongosh
- Also to perform connection from MongoDB to node.js driver connection string

**Step 1:** Install MongoDB First, you need to install MongoDB on your machine. You can download it from the official MongoDB website. After downloading, follow the instructions to install it.

**Step 2:** Start MongoDB Server You can start the MongoDB server by running the mongod command in your terminal. This will start the MongoDB server on the default port 27017.

```
C:\Users\DELL>mongod
{"t":{"$date":"2024-04-16T19:06:30.084+05:30"},"s":"I",
  "d wire specification","attr":{"spec":{"incomingExternalCl
  lClient":{"minWireVersion":0,"maxWireVersion":21},"outgoi
  :true}}}
```

**Step-3:** Connect to MongoDB Server using Mongoshell Open a new terminal window and connect to the MongoDB server using the mongo command. This will start the MongoDB shell (mongosh).

```
C:\Users\DELL>mongosh
Current Mongosh Log ID: 661e7f4e6dcde486d7117b7a
Connecting to:      mongodb://127.0.0.1:27017/?directConnectio
.2.4
Using MongoDB:      7.0.8
Using Mongosh:      2.2.4

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

-----
  The server generated these startup warnings when booting
  2024-04-16T18:47:26.981+05:30: Access control is not enabled fo
  iguration is unrestricted
-----
test>
```

Step – 4: Create the DataBase named the Student in the mongosh shell using the following command . Before creating the database first establish the connection with the MongoDB compass and place the localhost address in the mongoshell .

NAME:  
Roll No:

```
Please enter a MongoDB connection string (Default: mongodb://localhost/): mongodb://localhost:27017
mongodb://localhost:27017
Current Mongosh Log ID: 661e7bd148573ec585117b7a
Connecting to:      mongodb://localhost:27017/?directConnection=true&serverSelectionTimeoutMS=2000&
2.2.4
Using MongoDB:      7.0.8
Using Mongosh:      2.2.4

For mongosh info see: https://docs.mongodb.com/mongosh-shell/

-----
  The server generated these startup warnings when booting
  2024-04-16T18:47:26.981+05:30: Access control is not enabled for the database. Read and write access
  figuration is unrestricted
  -----
```

```
neeraj> use Student
switched to db Student
Student>
```

Step – 5: Insert the values into the documents under the database named Student:

```
Student> db.details.insertOne({name:"neeraj",age:"18"})
{
  acknowledged: true,
  insertedId: ObjectId('661e838748573ec585117b7c')
}
```

```
Student> db.details.insertMany([
... {
...   name:"Tayyab",
...   age:19
... },
... {
...   name:"Roshan",
...   age:19
... },
... {
...   name:"Sathish",
...   age:19
... }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('661e850048573ec585117b80'),
    '1': ObjectId('661e850048573ec585117b81'),
    '2': ObjectId('661e850048573ec585117b82')
  }
}
```



**NAME:**

**Roll No:**

**Step-6 :** Update the values in the documents named details under the database Student.

```
Student> db.details.updateOne({name:"neeraj"},{$set:{age:20}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

**Step- 7 :** Delete the any values in the details in the Database named Student:

```
Student> db.details.deleteOne({name:"neeraj"})
{ acknowledged: true, deletedCount: 1 }
Student> _
```

**Step – 8:** Connect the MongoDB server to Nodejs Application , first you need to install the mongodb in the nodejs using the npm

```
D:\nvn18>npm install mongodb
added 12 packages, and audited 82 packages in 13s

12 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```
D:\nvn18>npm install mongoose
added 8 packages, and audited 90 packages in 6s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

you can use the MongoClient object to connect to your MongoDB server:

- 1) Create the mongoose.js application and write the following code
- 2) Make sure that the mongoserver is running and make the connections
- 3) Print the all the values in the database.

NAME:

Roll No:

```
JS mongo.js > ...
1  const mongoose = require('mongoose');
2  const uri = 'mongodb://localhost:27017/Student';
3  mongoose.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true });
4  const db = mongoose.connection;
5  db.on('error', console.error.bind(console, 'MongoDB connection error:'));
6  db.once('open', async () => {
7    console.log('Connected to MongoDB');
8    const detailsSchema = new mongoose.Schema({
9      name: String,
10     age: Number,
11   });
12   const Details = mongoose.model('Details', detailsSchema);
13   try {
14     const docs = await Details.find({});
15     console.log(docs);
16   } catch (error) {
17     console.error('Error fetching documents:', error);
18   } finally {
19     mongoose.connection.close();
20   }
21 });
22
```

```
D:\nvn18>node mongo.js
Connected to MongoDB
[
  {
    _id: new ObjectId('661e850048573ec585117b80'),
    name: 'Tayyab',
    age: 19
  },
  {
    _id: new ObjectId('661e850048573ec585117b81'),
    name: 'Roshan',
    age: 19
  },
  {
    _id: new ObjectId('661e850048573ec585117b82'),
    name: 'Sathish',
    age: 19
  },
  {
    _id: new ObjectId('661e855d48573ec585117b83'),
    name: 'Neeraj',
    age: 18
  }
]
```