FULL STACK DEVELOPMENT- CS325

(LAB RECORD)

RVR & JC COLLEGE OF ENGINEERING

CHOWDAVARAM, GUNTUR-522 019 (AUTONOMOUS)

Affiliated to ACHARYA NAGARJUNA UNIVERSITY :: GUNTUR -10

Ch.Ratna Babu
Lecturer Incharge

Dr.M.Sreelatha
Prof.&HOD,Dept.of CSE

By :
D. Uday Sriram
(Y21CS031)

CS325-1(JOEL02)::LBD Course R-20 ::

FULL STACK DEVELOPMENT

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

- Steps for installation of Node.js environment Node
- Test through the node REPL shell commands
- Also install prompt-sync module using npm utility.
- Test and check the prompt-sync with console Module Application

Program:

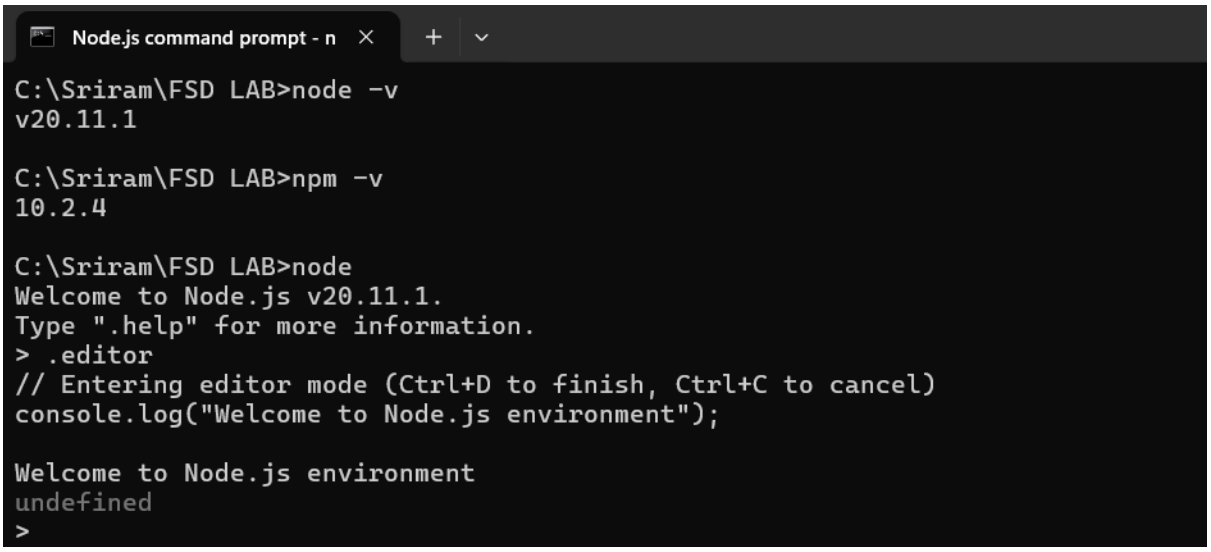
Steps for installation of Node.js environment:

1. Download the Node.js Installer: Visit the official Node.js website https://nodejs.org/download/ and download `.msi` installer.

2. Run the Node.js Installer: Open the downloaded `.msi` file. If the system prompts for, `Do you want to allow this app to make changes to your device? `, click `Yes`. The Node.js Setup wizard will open. Follow the on-screen instructions.

3. Verify the Installation: After the installation, you can verify that Node.js was properly installed by opening your command prompt or Windows PowerShell and running the following command: `node -v`.

Test through the node REPL shell commands:



```
C:\Sriram\FSD LAB>node -v
v20.11.1

C:\Sriram\FSD LAB>npm -v
10.2.4

C:\Sriram\FSD LAB>node
Welcome to Node.js v20.11.1.
Type ".help" for more information.
> .editor
// Entering editor mode (Ctrl+D to finish, Ctrl+C to cancel)
console.log("Welcome to Node.js environment");

Welcome to Node.js environment
undefined
>
```

Install the prompt-sync module:

Open your terminal and navigate to your project directory. Then, run the following command to install the prompt-sync module:

- ➢ npm install prompt-sync

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

1.Create a JavaScript file: Create a new JavaScript file in your project directory (for example, App.js). In this file, you can require the prompt-sync module and use it to get input from the user.

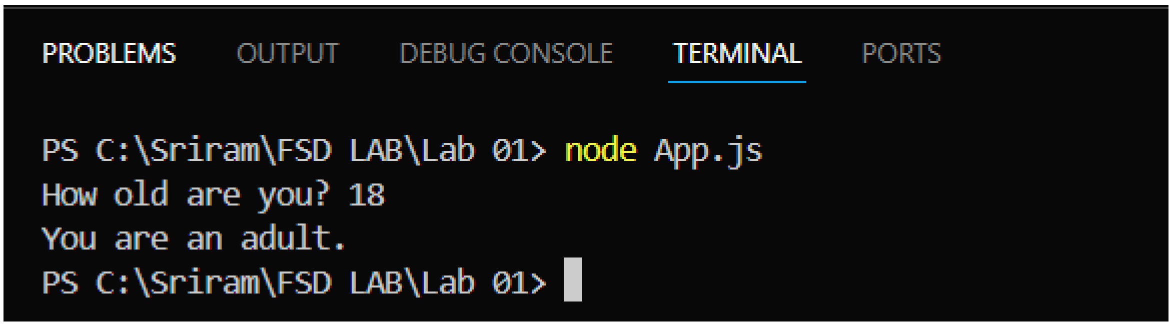
App.js:

```
var prompt = require('prompt-sync')();
var age = prompt('How old are you? ');
if (age < 18) {
   console.log('You are a minor.');
} else {
   console.log('You are an adult.');
}
```

2.Run the JavaScript file: In your terminal, run the JavaScript file using Node.js:

- node App.js

Output:



```
PS C:\Sriram\FSD LAB\Lab 01> node App.js
How old are you? 18
You are an adult.
PS C:\Sriram\FSD LAB\Lab 01>
```

1. **Aim:** 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.
   - Create Node Package Module myDate() using node utilities without package.json file
   - Also Create the Node Package Module myDate() using with package.json file directives like version,name,bin,etc.,
   - Also install created packaged module using npm utility

Program:

Create Node Package Module myDate() using node utilities without package.json file:

1.Create a JavaScript file:

Create a new JavaScript file in your project directory (for example, myDate.js). In this file, you can export a function that returns the current date and time.

mydate.js:

```
exports.myDate = function() {
  return new Date();
};
```

2.Test the myDate module:

Create another JavaScript file (for example, date.js). In this file, you can require the myDate module and use it to print the current date and time.

date.js:

```
var myDate = require('./myDate');
console.log(“Today Date : “+myDate.myDate());
```

3.Run the JavaScript file:

In your terminal, run the JavaScript file using Node.js

- ➢ node app.js

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

1.Create a JavaScript file:

Create a new JavaScript file in your project directory (for example, myDate.js). In this file, you can export a function that returns the current date and time.

mydate.js:

```
exports.myDate = function() {
   return new Date();
};
```

2.Test the myDate module:

Create another JavaScript file (for example, date.js). In this file, you can require the myDate module and use it to print the current date and time.

date.js:

```
var myDate = require('./myDate');
console.log("Today Date : "+myDate.myDate());
```

3.Create a JSON file:

Create a new JSON file in your project directory (package.json). In this file, you can add name, version, author, description etc.. of your module.

package.json:

```
{
   "name": "mydateus",
   "version": "1.0.0",
   "description": "A simple module that returns the current date and time",
   "main": "myDate.js",

   "keywords": [
     "date",
     "time"
   ],
   "author": "Sriram",
 }
```

Also install created packaged module using npm utility:

1.Publish the module:

Publish the module into npm registry using the following command.

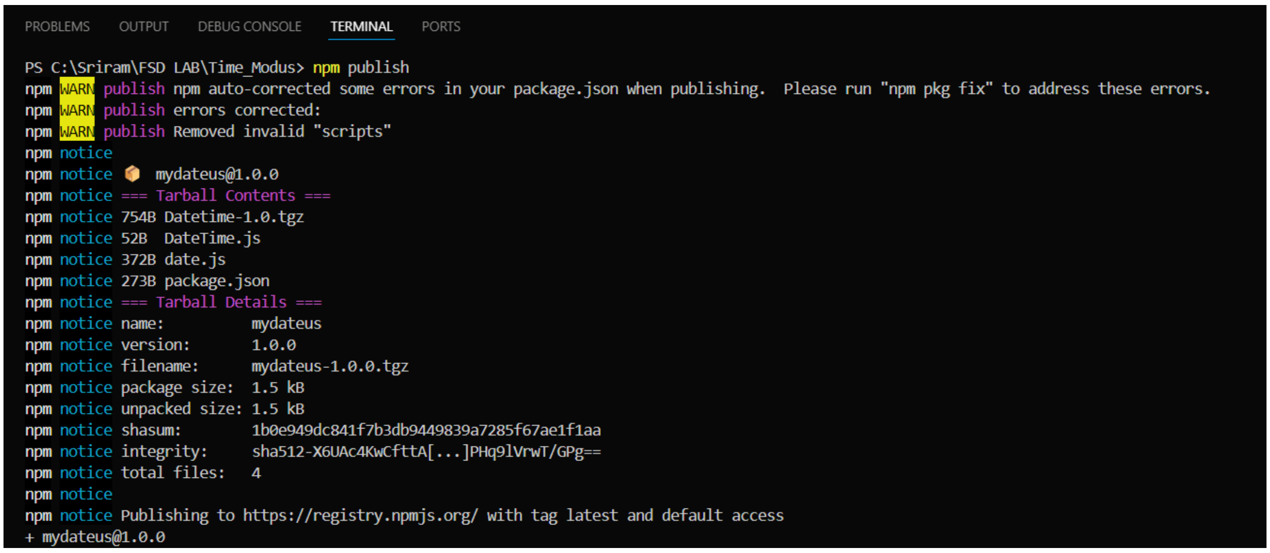
- npm publish

2.Installing the module:

Use the following command to install the created module(mydateus).

- npm install mydateus

Output:



```
PS C:\Sriram\FSD LAB\Time_Modus> npm publish
npm WARN publish npm auto-corrected some errors in your package.json when publishing.  Please run "npm pkg fix" to address these errors.
npm WARN publish errors corrected:
npm WARN publish Removed invalid "scripts"
npm notice
npm notice 📦  mydateus@1.0.0
npm notice === Tarball Contents ===
npm notice 754B Datetime-1.0.tgz
npm notice 52B  DateTime.js
npm notice 372B date.js
npm notice 273B package.json
npm notice === Tarball Details ===
npm notice name:          mydateus
npm notice version:       1.0.0
npm notice filename:      mydateus-1.0.0.tgz
npm notice package size:  1.5 kB
npm notice unpacked size: 1.5 kB
npm notice shasum:        1b0e949dc841f7b3db9449839a7285f67ae1f1aa
npm notice integrity:     sha512-X6UAc4KwCfttA[...]PHq9lVrwT/GPg==
npm notice total files:   4
npm notice
npm notice Publishing to https://registry.npmjs.org/ with tag latest and default access
+ mydateus@1.0.0
```

2. **Aim:** Create Node JS Application with Folder structure using npm utilities and develop one application to display "welcome Node JS APP" Greet message
   - With VisualStudioCode APP Framework(Any other)
   - Without VisualStudioCode APP Framework
   - Also Access the Custom myDate Module.

Program:

1. Create a new directory:

Create a new directory for your project using following command in your terminal

```
mkdir myNodeApp
```

2.Navigate into that directory:

Navigate into your project directory using command

```
cd myNodeApp
```

3. Create a new folder named modules:

Inside the modules folder, create a new file named Datetime.js using following commands

```
mkdir modules
cd modules && notepad Datetime.js
```

Datetime.js:

```
exports.Datetime = function () {
  return Date();
};
```

4. Navigate back to root directory using following command:

```
cd..
```

5. Create a new file named app.js

Create a new js file in your root directory using command

```
notepad app.js
```

app.js:

```
var http = require('http');
var date = require('./modules/DateTime');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write("<h1 style='color:blue;'>Welcome Node JS APP</h1>");
  res.write("<h1 style='color:black;'>Current date and time are: <span
style='color:red;'>" + date.Datetime() + "</span></h1>");
```

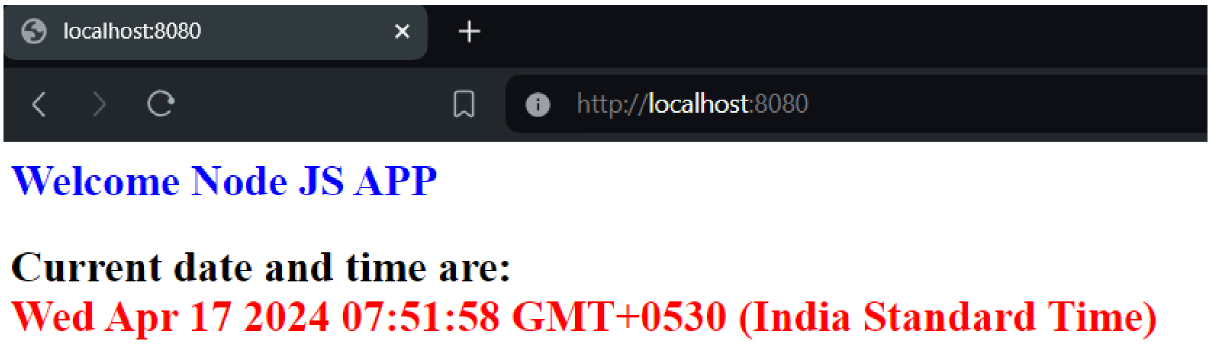
```
  res.end();
}).listen(8080);
```

6. Run the application:

Run the Node JS Application using the following command

- node app.js

Output:



Welcome Node JS APP

Current date and time are:
Wed Apr 17 2024 07:51:58 GMT+0530 (India Standard Time)

3. **Aim:** Create Angular CLI Applications with different component configuration steps using different @Angular ng module utilities at CLI environment.
   - Class component Angular app
   - Define Inline selector component in Angular HelloWorld app with root element
   - Define Inline template component in Angular HelloWorld app with HTML elements

Program:

Class component Angular app:

1.Install Angular CLI:

```
npm install -g @angular/cli
```

2.Create a new Angular project:

```
ng new HelloWorld - -no-standalone
```

3.Navigate into your new project:

```
cd HelloWorld
```

4.Generate a new component:

```
ng generate component mycomponent
```

In the app.component.ts file, you can define a class component like this:

App.component.ts:

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
@Component({
 selector: 'app-root',
 standalone: true,
 imports: [RouterOutlet],
 templateUrl: './mycomponent/mycomponent.component.html',
 styleUrl: './mycomponent/mycomponent.component.css'
})
export class AppComponent {
 title = 'HelloWorld';
}
```

Define Inline selector component in Angular HelloWorld app with root element:

You can define an inline selector in the @Component decorator:

In mycomponent.component.ts file update this:

Mycomponent.component.ts:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-mycomponent',
  templateUrl: './mycomponent.component.html',
  styleUrl: './mycomponent.component.css'
})
export class MycomponentComponent {
}
```

Chech wheter MycomponentComputer class is added in app.module.ts or not

If not add using following

```
import { MycomponentComponent } from './mycomponent/mycomponent.component';
@NgModule({
  declarations: [
    AppComponent,
    MycomponentComponent
  ],
```

Now update app.component.html file:

```
<div>
  <app-mycomponent></app-mycomponent> (this is selector of our custom selector)
</div>
```

Define Inline template component in Angular HelloWorld app with HTML elements

Defining an inline template in the @Component decorator:

app.component.ts:

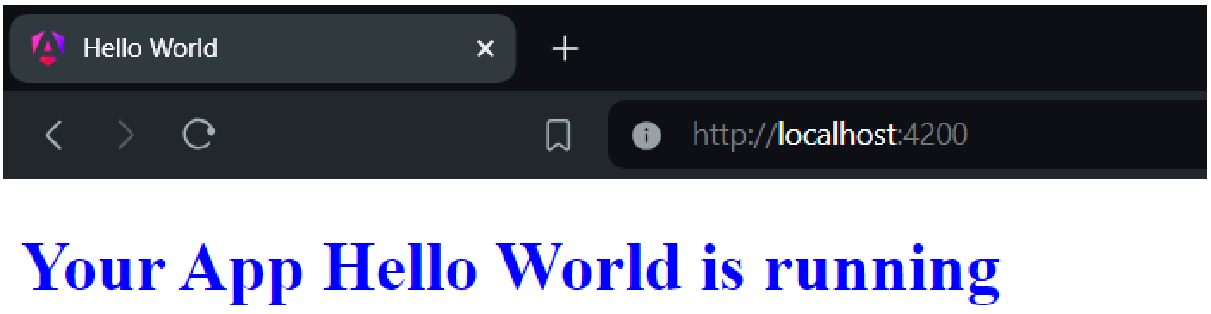
```
@Component({
  selector: 'app-my-component',
  template: `<h1>{{title}}</h1>`,
  styleUrls: ['./my-component.component.css'] })
```

Define Inline Style component in Angular HelloWorld app to style the color of the message:

Defining inline styles in the @Component decorator:

```
@Component({
  selector: 'app-my-component',
 template: `<h1>{{title}}</h1>`,
 styles: ['h1 { color: red; }']
})
```

Output:



Your App Hello World is running

4. **Aim:** Create Angular CLI Applications using Angular Class component constructors and objects and different variable initialization.
   - Create Date Class Constructor with current Date in Class Component
   - By using Selector,templateURL and styleURL External component configurations demonstrate the constructor with different objects

Program:

Create Date Class Constructor with current Date in Class Component:

1.Install Angular CLI:

```
npm install -g @angular/cli
```

2.Create a new Angular project:

```
ng new Date - -no-standalone
```

3.Navigate into your new project:

```
cd Date
```

4.Generate a new component:

```
ng generate component datecomponent
```

5.Update datecomponent.component.ts:

<u>Datecomponent.component.ts:</u>

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-datecomponent',
  templateUrl: './datecomponent.component.html',
  styleUrl: './datecomponent.component.css'
})
export class DatecomponentComponent {
  currentDate: Date;
  constructor() {
    this.currentDate = new Date();
  }
}
```

By using Selector,templateURL and styleURL External component configurations demonstrate the constructor with different objects:

1.Update datecomponent.component.html:

datecomponent.component.html:

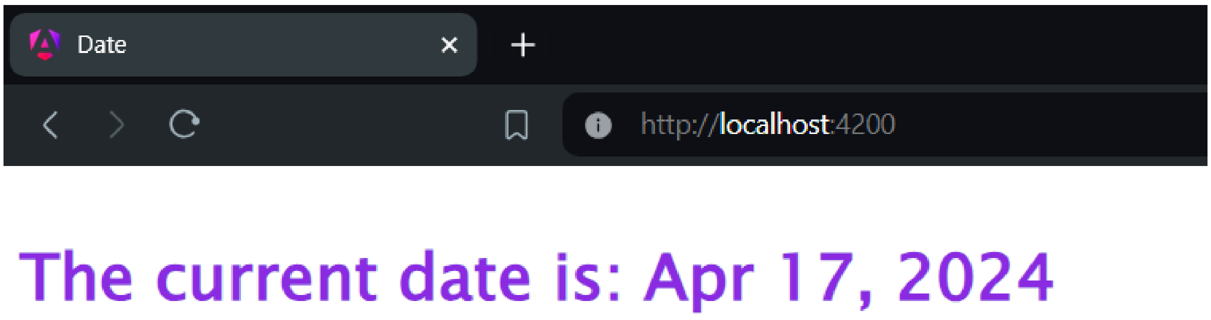
```
<p>The current date is: {{ currentDate | date }}</p>
```

2. Update datecomponent.component.css:

datecomponent.component.css:

```
p{
   color:blueviolet;
   font-family: 'Lucida Sans', 'Lucida Sans Regular', 'Lucida Grande', 'Lucida Sans Unicode',
Geneva, Verdana, sans-serif;
}
```

Output:



```
The current date is: Apr 17, 2024
```

5. **Aim:** 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

Program:

Create different Angular Expressions in Class Component:

1.Install Angular CLI:

```
npm install -g @angular/cli
```

2.Create a new Angular project:

```
ng new expr - -no-standalone
```

3.Navigate into your new project:

```
cd expr
```

4.Update app.component.html:

app.component.html:

```
<html>
<body>
<h1>Expressions</h1>
 Number:<br>
 {{5}}<hr>
 String:<br>
3 | P a g e
 {{'My String'}}<hr>
 Adding two strings together:<br>
 {{'String1' + ' ' + 'String2'}}<hr>
 Adding two numbers together:<br>
 {{5+5}}<hr>
 Adding strings and numbers together:<br>
 {{5 + '+' + 5 + '='}}{{5+5}}<hr>
 Comparing two numbers with each other:<br>
 {{5===5}}<hr>
</body>
</html>
```

Also Specify with Different Angular pipes or filters to demonstrate each filter with Angular expression:

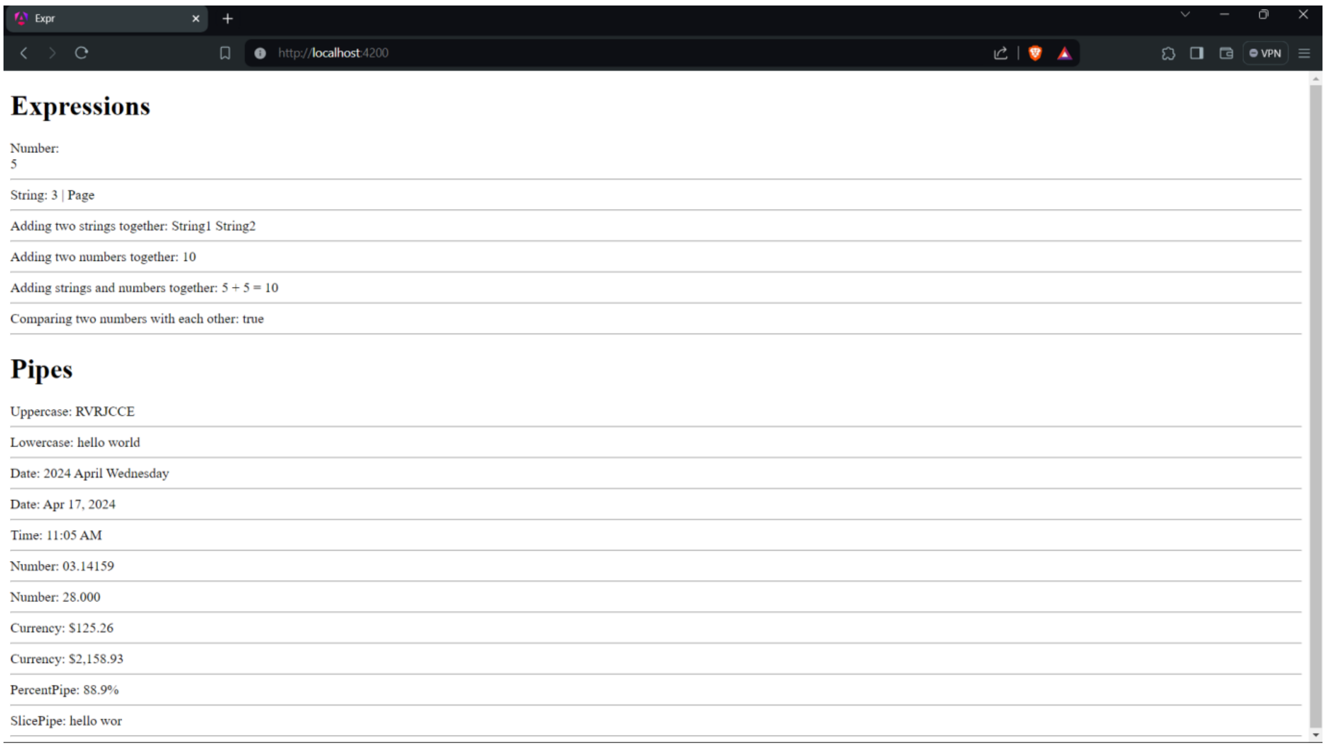
1.Update app.component.html:

```
<html>

<body>

<h1>Pipes</h1>

Uppercase: {{ 'rvrjcce' | uppercase }}<hr>

Lowercase: {{ 'HELLO WORLD' | lowercase }}<hr>

Date: {{ today | date:'y MMMM EEEE' }}<hr>

Date: {{ today | date:'mediumDate' }}<hr>

Time: {{ today | date:'shortTime' }}<hr>

Number: {{ 3.1415927 | number:'2.1-5' }}<hr>

Number: {{ 28 | number:'2.3' }}<hr>

Currency: {{ 125.257 | currency:'USD':true:'1.2-2' }}<hr>

Currency: {{ 2158.925 | currency }}<hr>

PercentPipe: {{ 0.8888 | percent:'2.1' }}<hr>

SlicePipe: {{ 'hello world' | slice:0:9 }}<hr>

</body>

</html>
```

2.Run the app using following command:

- ng serve

Output:



```
Expressions

Number:
5
String: 3 | Page
Adding two strings together: String1 String2
Adding two numbers together: 10
Adding strings and numbers together: 5 + 5 = 10
Comparing two numbers with each other: true

Pipes

Uppercase: RVRJCCE
Lowercase: hello world
Date: 2024 April Wednesday
Date: Apr 17, 2024
Time: 11:05 AM
Number: 03.14159
Number: 28.000
Currency: $125.26
Currency: $2,158.93
PercentPipe: 88.9%
SlicePipe: hello wor
```

6. **Aim:** Create Angular CLI Applications using Data Binding demonstrate each binding type with form elements.
   - Interpolation Binding.
   - Style Binding
   - Class Binding.
   - Two –way binding

Program:

Create a new Angular CLI application:

To create a new Angular CLI application, you can use the following command in your terminal or command prompt

- ng new my-app

Create a new component:

To create a new component, you can use the following command in your terminal or command prompt:

- ng generate component mycomponent

Interpolation Binding:

Interpolation binding is used to display data from the component to the view (DOM). It is denoted by double curly braces {{ }}.

mycomponent.component.ts:

```
export class MyComponent {
  title = 'My Component';
}
```

In my-component.component.html:

```
<h1>{{ title }}</h1>
```

Style Binding:

Style binding is used to bind the data from the component to the HTML style property. It is denoted by [style.property].

mycomponent.component.ts:

```
export class MyComponent {
  backgroundColor = 'red';
}
```

mycomponent.component.html:

```
<div [style.backgroundColor]="backgroundColor">This is a red div</div>
```

Class Binding:

Class binding is used to bind the data from the component to the HTML class property. It is denoted by [class.class-name]. Here's an example:In my-component.component.ts:

```
export class MyComponent {
  isActive = true;
}
```

my-component.component.html:

```
<div [class.active]="isActive">This is an active div</div>
```

Two-Way Binding:

Two-way binding is used to update the component property when the user interacts with the element. In this example, we use two-way binding with ngModel to update the name property when the user types in the input field:

mycomponent.component.html:

```
<div>
   <input [value]="firstname" [(ngModel)]="firstname">
</div>
<div>Hello{{ firstname }}</div>
```

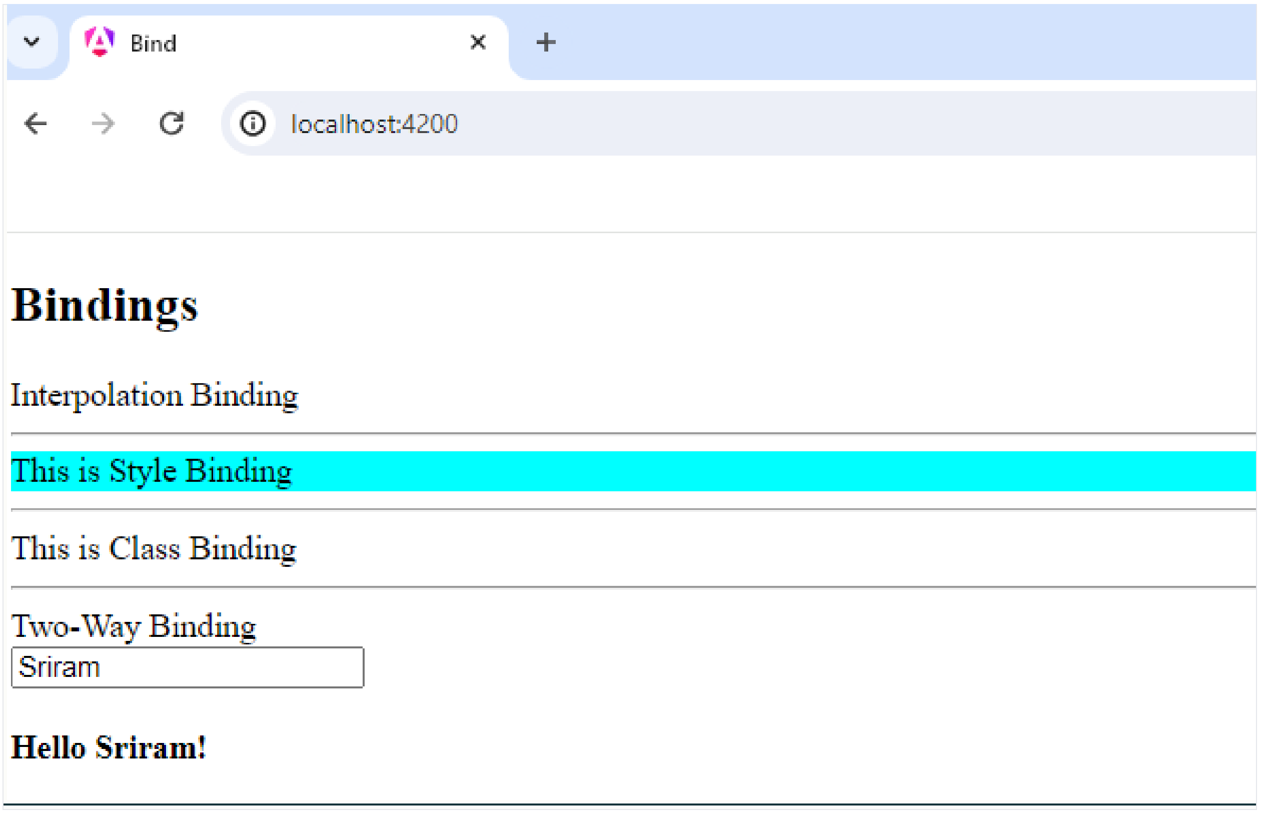
mycomponent.component.ts:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-mycomponent',
  templateUrl: './mycomponent.component.html',
  styleUrl: './mycomponent.component.css'
})
export class MycomponentComponent {
  firstname: string = "Sriram"; // Ensure firstname is initialized to an empty string
}
```

Run the app using following command:

- ng serve

Output:



Bindings

Interpolation Binding

This is Style Binding

This is Class Binding

Two-Way Binding

Sriram

Hello Sriram!

8.Aim: 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();

Program:

Create a Node.js file using command:

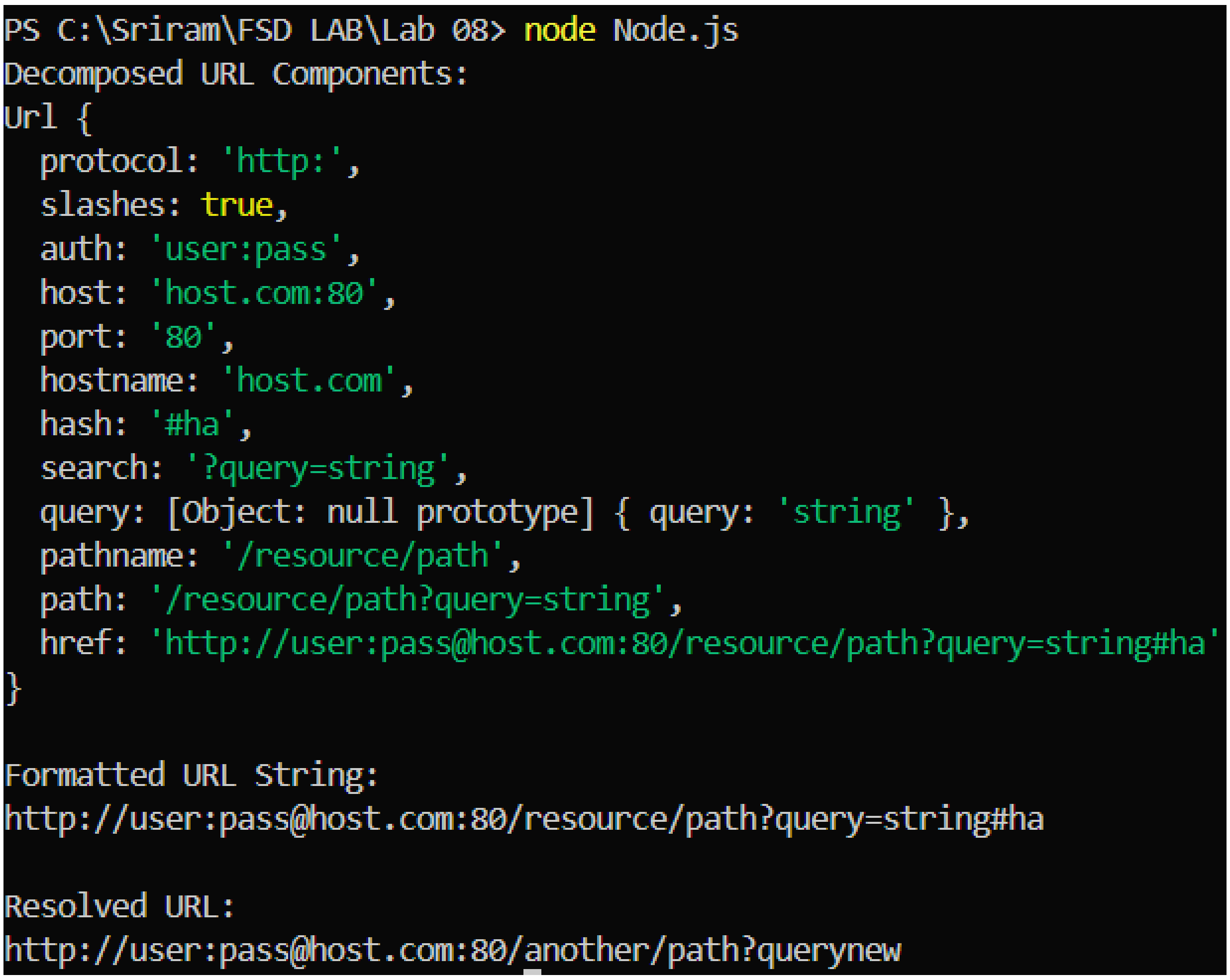
- notepad Node.js

```
const url = require('url');
let urlStr = 'http://user:pass@host.com:80/resource/path?query=string#ha';
console.log('Decomposed URL Components:');
let urlobj = url.parse(urlStr, true);
console.log(urlobj);
console.log('\nFormatted URL String:');
let formattedUrlStr = url.format(urlobj);
console.log(formattedUrlStr);
console.log('\nResolved URL:');
let baseUrl ='http://user:pass@host.com:80/resource/path?query=string#ha';
let targetUrl = '/another/path?querynew';
let resolvedUrl = url.resolve(baseUrl, targetUrl);
console.log(resolvedUrl);
```

Run Node.js file using command:

- node Node.js

Output:



```
PS C:\Sriram\FSD LAB\Lab 08> node Node.js
Decomposed URL Components:
Url {
  protocol: 'http:',
  slashes: true,
  auth: 'user:pass',
  host: 'host.com:80',
  port: '80',
  hostname: 'host.com',
  hash: '#ha',
  search: '?query=string',
  query: [Object: null prototype] { query: 'string' },
  pathname: '/resource/path',
  path: '/resource/path?query=string',
  href: 'http://user:pass@host.com:80/resource/path?query=string#ha'
}

Formatted URL String:
http://user:pass@host.com:80/resource/path?query=string#ha

Resolved URL:
http://user:pass@host.com:80/another/path?querynew
```

9.Aim: 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

Program:

Create Http Static server files data using static files:

1.Create the Directory Structure:

Create directories named http and html using following commands

```
mkdir http && mkdir html
```

2.Write the Server Code:

1.Navigate to the http directory and Create a server.js file

```
cd http && notepad server.js
```

server.js:

```
var fs = require('fs');
var http = require('http');
var url = require('url');
var path = require('path');
var ROOT_DIR = "C:\\Sriram\\FSD LAB\\Lab 09\\html\\";
http.createServer(function (req, res) {
  var urlObj = url.parse(req.url, true, false);
  var filePath = path.join(ROOT_DIR, urlObj.pathname);
  if (fs.statSync(filePath).isDirectory()) {
    filePath = path.join(filePath, 'index.html');
  }
  fs.readFile(filePath, function (err, data) {
    if (err) {
      res.writeHead(404);
      res.end(JSON.stringify(err));
      return;
    }
    res.writeHead(200);
    res.end(data);
});
}).listen(8095);
console.log('Server running at http://127.0.0.1:8095/');
```

Define HttpRequest/HttpResponse objects:

Write the Client Code:

1.Navigate to the http directory using following command

- ➢ cd http

2.Create a client.js file using command

- ➢ notepad client.js

client.js:

```
var http = require('http');
var options = {
   hostname: '218.248.4.102',
   port: 8095,
   path: '/index.html'
};
function handleResponse(response) {
   var serverData = '';
   response.on('data', function (chunk) {
      serverData += chunk;
   });
   response.on('end', function () {
      console.log(serverData);
   });
}
var req = http.request(options, function(response) {
   handleResponse(response);
});
req.end();
```

Write the HTML Code:

1.Navigate to the html directory using following command

- cd html

2.Create a index.html file using command

- notepad index.html

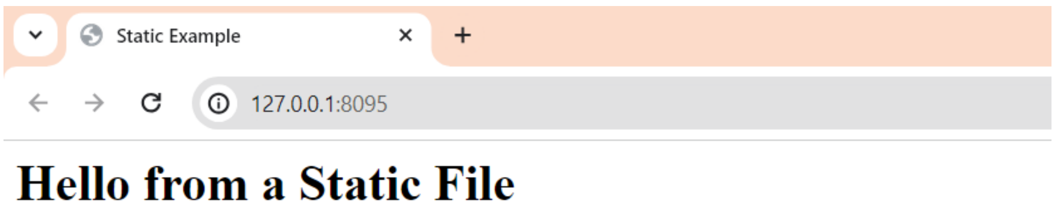
index.html:

```
<html>
<head>
<title>Static Example</title>
</head>
<body>
<h1>Hello from a Static File</h1>
</body>
</html>
```

Run the js files in separate command prompts using commands:

- node client.js
- node server.js

Output:



Hello from a Static File

10.Aim: 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

Program:

Create a HTML file using command

- notepad index.html

index.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>Arithmetic Operations</title>
</head>
<body>
  <form action="/add" method="post">
    <label for="n1">Number 1:</label>
    <input type="text" id="n1" name="n1"><br><br>

    <label for="n2">Number 2:</label>
    <input type="text" id="n2" name="n2"><br><br>

    <button type="submit">ADD</button>
    <button type="reset">Reset</button>
  </form>
</body>
</html>
```

Create a express.js file using command:

- notepad express.js

Install express using command:

- npm install express

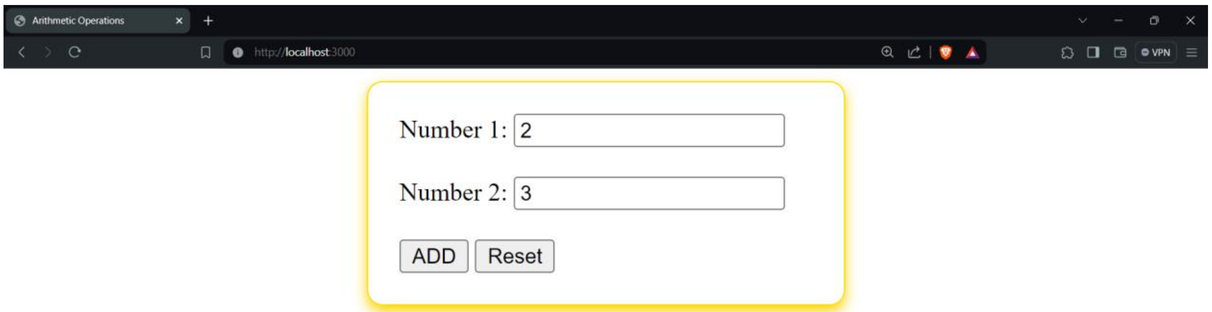
express.js:

```
const express = require('express');
const app = express();
const port = 3000;
app.use(express.urlencoded({ extended: true }));
app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});
app.post('/add', (req, res) => {
  const num1 = parseInt(req.body.n1);
  const num2 = parseInt(req.body.n2);
  const result = num1 + num2;
    res.send(`
    <h1>Result: ${result}</h1>
    <a href="http://localhost:3000/">Redirect back</a>
  `);
});
app.get('/http://localhost:3000/', (req, res) => {
  res.send('');
});
app.listen(port, () => {
  console.log(`Server running at http://localhost:${port}`);
});
```

Run the express.js using command:

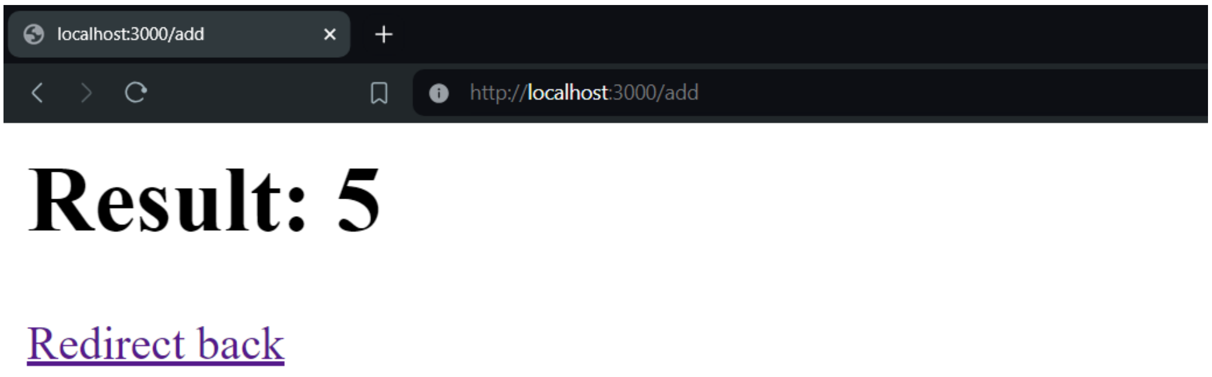
- node express.js

Output:



Number 1: 2

Number 2: 3



Result: 5

Redirect back

11.Aim: 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.

Program:

Create a express.js file using command:

- ➢ notepad express.js

Install express using command:

- ➢ npm install express

express.js:

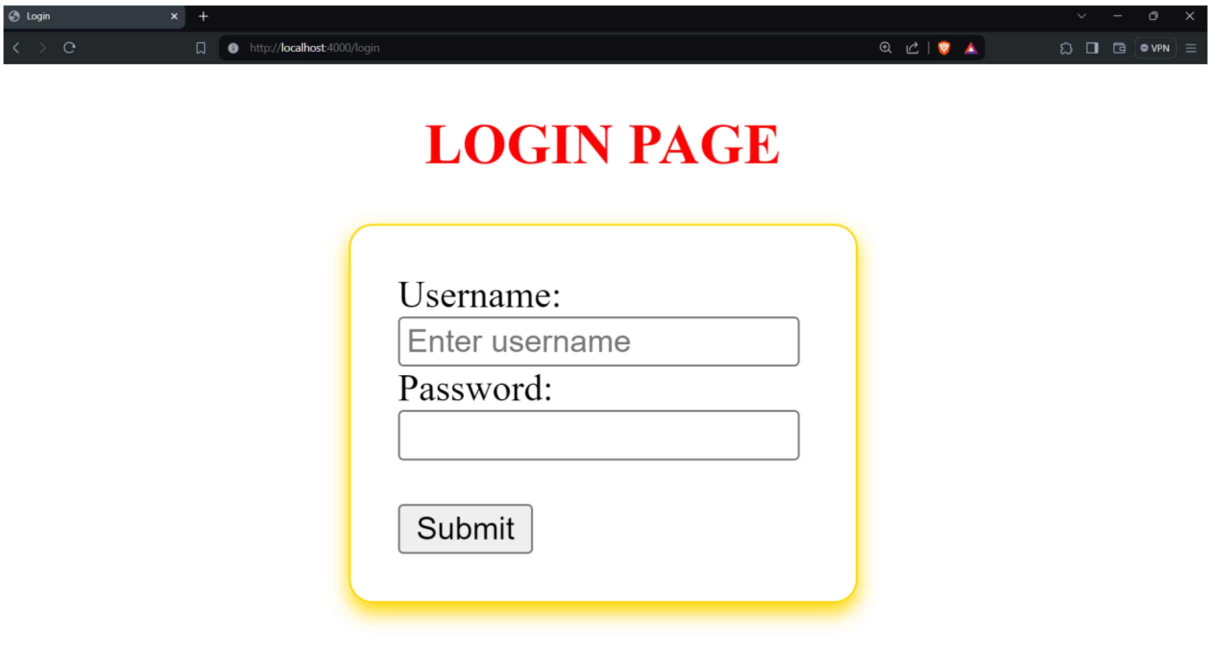
```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
app.use(bodyParser.urlencoded({ extended: true }));
app.get('/login', (req, res) => {
  res.send(`
    <!DOCTYPE html>
    <html>
    <head>
      <title>Login</title>
      <style>
        h2 {
          text-align: center;
          margin-bottom: 20px;
          color:red;
        }
      </style>
    </head>
    <body>
    <h2>LOGIN PAGE</h2>
```

```
      <form action="/login" method="post">
        <label for="username">Username:</label><br>
        <input type="text" id="username" name="username" placeholder="Enter
username"><br>
        <label for="password">Password:</label><br>
        <input type="password" id="password" name="password"><br><br>
        <input type="submit" value="Submit">
      </form>
    </body>
    </html>
  `);
});
app.post('/login', (req, res) => {
  const username = req.body.username;
  const password = req.body.password;

  // Replace 'admin' and 'password' with your own validation logic
  if(username === 'y21cs031' && password === 'y21cs031') {
    res.send('Login successful!');
  } else {
    res.send('Invalid login credentials.');
  }
});
app.listen(4000, () => {
  console.log('Server is running on port http://localhost:4000/login');
});
```

Run the express.js using command:

- node express.js

Output:



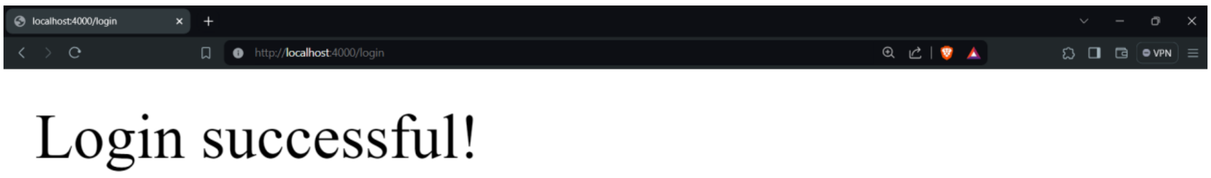
LOGIN PAGE

Username:

Enter username

Password:

Submit



Login successful!

12.Aim: Create Simple MongDB 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

Program:

MongoDB Installation:

1. Install MongoDb Community edition from the link :

https://www.mongodb.com/try/download/community

2. Install MongoShell from the link :

https://www.mongodb.com/try/download/shell

3. Open MongoDb Compass,start the server

4. Open Mongoshell application and paste the connection string url there

Create a simple student document database:

Create a new database using command

- ➢ use student

Insert one student record in mongosh:

Insert a new document into the 'students' collection using command

- ➢ db.student.insertOne({name:"Uday Sriram",id:"Y21CS031",branch:"Computer Science"})

Update and delete one document in mongosh:

Updating name of inserted document using command

- ➢ db.student.update({id:"Y21CS031"},{$set:{name:"Uday Sriram Dutta"}})

Deleting inserted document using command

- ➢ db.student.remove({id:"Y21CS031"})

Perform connection from MongoDB to Node.js driver connection string:

First, install the MongoDB driver in your Node.js project using command

- ➢ npm install mongodb

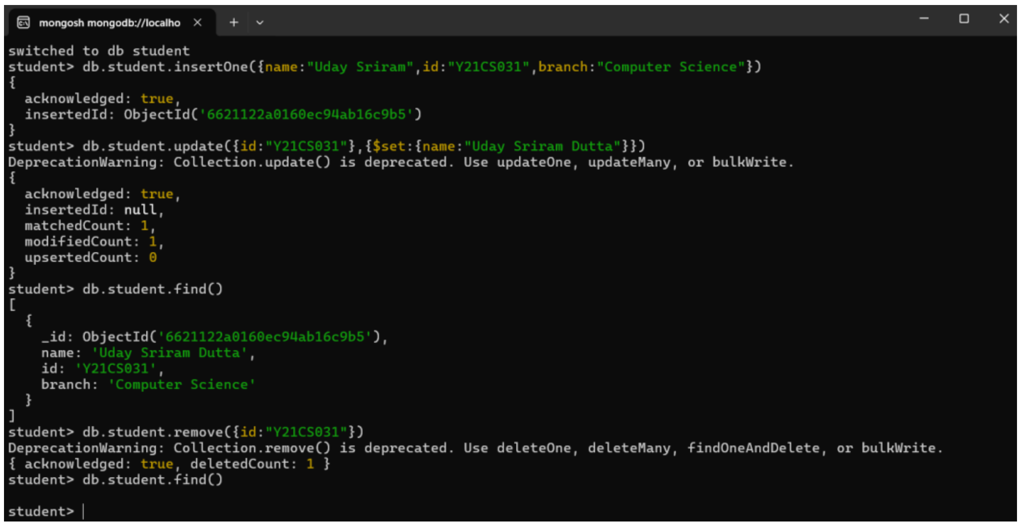
Node.js:

```
var MongoClient = require('mongodb').MongoClient;
var uri = "mongodb://localhost:27017";
const client = new MongoClient(uri);
client.connect();
console.log("Connected to MongoDB");
const database = client.db('test13');
const studentsCollection = database.collection('testc2');
const result = studentsCollection.insertOne({ name: "Uday Sriram",
id:"Y21CS031" , branch: "Computer Science" });
console.log(`Inserted document with ID: ${result.insertedId}`);
```

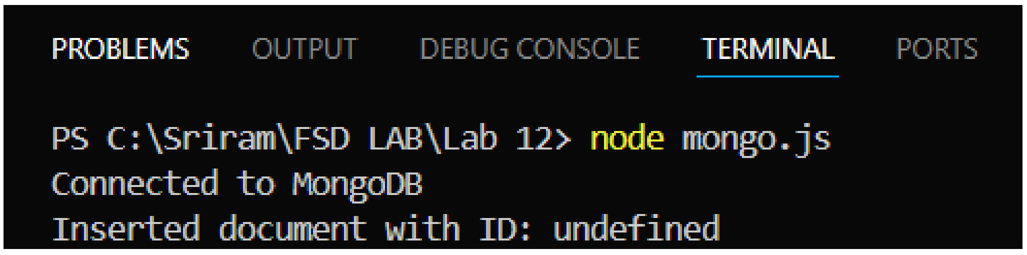
Run the Node.js file using command:

- node Node.js

Output:



```
switched to db student
student> db.student.insertOne({name:"Uday Sriram",id:"Y21CS031",branch:"Computer Science"})
{
  acknowledged: true,
  insertedId: ObjectId('6621122a0160ec94ab16c9b5')
}
student> db.student.update({id:"Y21CS031"},{$set:{name:"Uday Sriram Dutta"}})
DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or bulkWrite.
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
student> db.student.find()
[
  {
    _id: ObjectId('6621122a0160ec94ab16c9b5'),
    name: 'Uday Sriram Dutta',
    id: 'Y21CS031',
    branch: 'Computer Science'
  }
]
student> db.student.remove({id:"Y21CS031"})
DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMany, findOneAndDelete, or bulkWrite.
{ acknowledged: true, deletedCount: 1 }
student> db.student.find()

student> |
```



```
PS C:\Sriram\FSD LAB\Lab 12> node mongo.js
Connected to MongoDB
Inserted document with ID: undefined
```