

**Shri Ramdeobaba College of Engineering & Management  
Nagpur-13**

**Department of Computer Application**

**Session: 2023-2025**



**Submission for**

**Course Name:** Full Stack Web Development Using Mean

**Course Code:** MCP548

**Name of the Student:** Jayesh Lalit Nandanwar

**Class Roll No:** 26

**Semester:** MCA II Semester

**Shift:** 2

**Batch:** B2

Under the Guidance of  
Prof. Vishnu Vardhan

Date of submission: 27/04/2024

## INDEX

Sr. No.	TITLE	Page No.	Date	Remark
1	Creating basic web page designing using markup tags	3		
2	Implementing JavaScript functionality in a webpage by different ways.	5		
3	Creating rich interfaces using CSS on webpage by different ways.	7		
4	Installation of NodeJs and setting up of server using NodeJs.	12		
5	Knowing NodeJs REPL environment and basic Commands.	13		
6	Creating a Node Module and using NPM packages.	14		
7	Demonstrating the Asynchronous programming using node.js.	16		
8	Installation of Express and launching the server using ExpressJs.	17		
9	Implementing MVC pattern for Express for Express.js Application.	19		
10	Rendering of views and static files using Express.js.	23		
11	Implementing of web routes using express js.	26		
12	Using the cookies and session for persisting the user data.	29		
13	Installing MongoDB instance to run it for accessing through client.	32		
14	Creation of database and collections using MongoDB.	33		
15	Implementing the CRUD operations using MongoDB queries.	34		
16	Connecting Express.js webpages through Mongoose for MongoDB.	37		
17	Creating AngularJs modules for using in front-end web pages.	39		
18	Demonstrate AngularJS Modules and Directives in an HTML page.	40		
19	Implementing Dependency Injection using AngularJs for a given webpage.	41		

## Practical 1

**Practical name:** Creating basic web page designing using markup tags.

Markup tags, also known as HTML tags, are elements used in Hypertext Markup Language (HTML) to structure and format the content of web pages. HTML tags are enclosed in angle brackets (< >) and consist of a tag name or keyword.

Markup tags are used to define the structure of a web page and provide instructions to web browsers on how to display and interpret the content. They indicate the beginning and end of specific elements or sections within an HTML document.

- <html>
- <head>
- <body>
- <h1> to <h6>
- <p>
- <a>
- <img>
- <ul>
- <ol>
- <li>
- <table>
- <tr>
- <td>
- <form> etc.

Are some markup tags.

**Code in which some markup tags are implemented:**

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Webpage</title>
  </head>
  <body>
    <h1>Welcome to My Webpage</h1>
    <h2>Sub-heading</h2>
    <h3>Sub-sub-heading</h3>
    <h4>And so on...</h4>
```

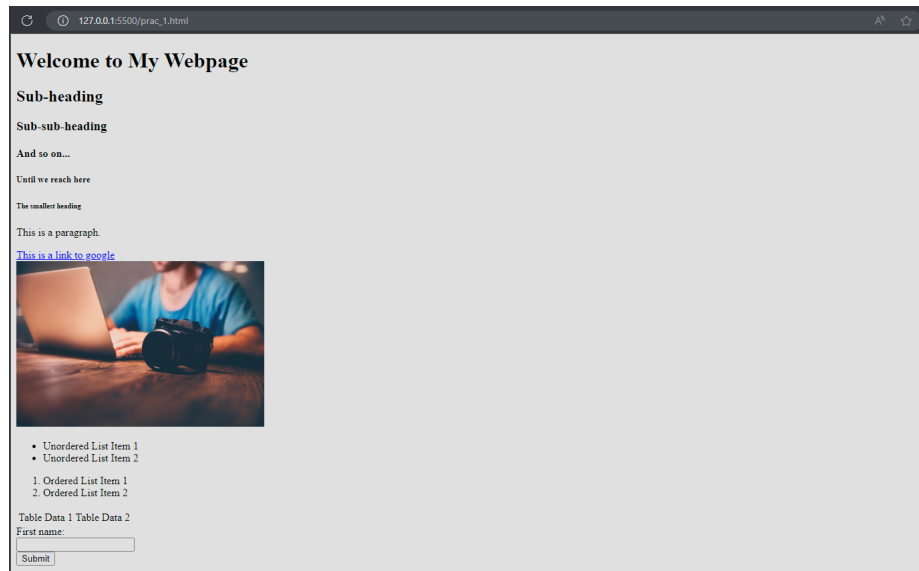
```

<h5>Until we reach here</h5>
<h6>The smallest heading</h6>
<p>This is a paragraph.</p>
<a href="https://www.google.com">This is a link to google</a><br>

<ul>
  <li>Unordered List Item 1</li>
  <li>Unordered List Item 2</li>
</ul>
<ol>
  <li>Ordered List Item 1</li>
  <li>Ordered List Item 2</li>
</ol>
<table>
  <tr>
    <td>Table Data 1</td>
    <td>Table Data 2</td>
  </tr>
</table>
<form action="/submit_form" method="post">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname"><br>
  <input type="submit" value="Submit">
</form>
</body>
</html>

```

**Output:**



## Practical 2

**Practical name:** Implementing JavaScript functionality in a webpage by different ways.

There are multiple ways to embed JavaScript functionality into a webpage. Here are some common methods:

1. **Inline JavaScript:** You can embed JavaScript code directly within the HTML document using the `<script>` tag. For example:  

```
<script>
// JavaScript code here alert(HI,there!);
</script>
```
2. **External JavaScript file:** You can create a separate .js file containing your JavaScript code and link it to your HTML document using the `<script>` tag with the `src` attribute. For example:  

```
<script src="(yourfilename).js"></script>
```

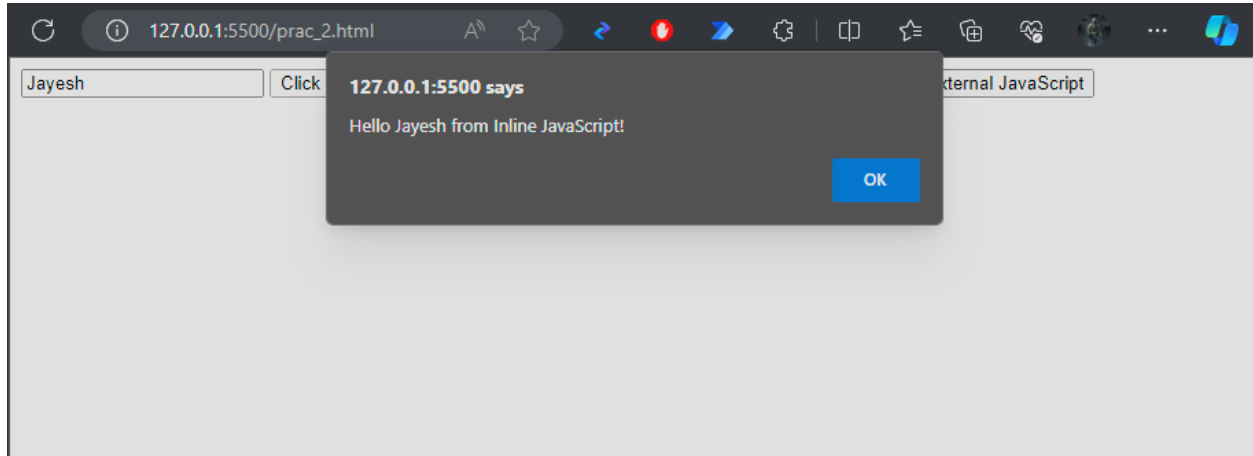
This approach is recommended for larger scripts or when you want to separate your JavaScript code for better organization and reusability.

Code in which javascript functionality is embedded-:

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Example</title>
  <input id="name" type="text" onchange="getName()">
  <script src="externaljs.js"></script>
</head>
<body>
  <button
    onclick="var name = document.getElementById('name').value;alert('Hello ' + name + ' from
Inline JavaScript!')">Click
    Me for Inline JavaScript</button>
  <button onclick="sayHelloInternal()">Click Me for Internal JavaScript</button>
  <button onclick="sayHelloExternal()">Click Me for External JavaScript</button>
</body>
<script>
  var name;
  function getName() {
    name = document.getElementById("name").value;
```

```
}  
function sayHelloInternal() {  
    alert('Hello ' + name + ' from Internal JavaScript!');  
}  
</script>  
</html>
```

### Output:



In this output the box appear in the top-center is an “Alert” It is often used for simple notifications or user prompts. The input is taken from input box and displayed in alert.

## Practical 3

**Practical name:** Creating rich interfaces using CSS on webpage by different ways.

There are several ways to apply CSS formatting to a webpage.

1. **Inline CSS:** You can apply CSS styles directly within HTML elements using the style attribute. For example:

```
<p style="color: blue; font-size: 16px;">This is a paragraph with inline styles.</p>
```

In this in-line css style is used to specify css in html code. This in-line css changes color&font size of words wwhich are written inside the <p> tag.

2. **Internal CSS:** You can include CSS styles within the <style> tags in the <head> section of the HTML document. For example:

```
<!DOCTYPE html>
<html>
<head>
  <title>CSS Example</title>
  <style> p {
    color: blue; font-size: 16px;
  }
</style>
</head>
<body>
  <p>This is a paragraph with internal CSS styles.</p>
</body>
</html>
```

In this internal css style is used to specify css in html code. This internal css changes color&font size of words which are written inside the <p> tag. This css can change the all <p> tag style present in the page.

3. **External CSS file:** You can create a separate .css file containing CSS styles and link it to your HTML document using the <link> tag. The CSS styles are stored in the styles.css file, which should be located in the same directory as your HTML file. This method is recommended for larger projects or when you want to separate the CSS code for better organization and reusability.

### Code in which External CSS is embedded:

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

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <link rel="stylesheet" href="style.css">
  <style type="text/css">
    @import url("http://google.com/styles/css1.css");

    * {
      color: grey;
    }

    h1,
    h2,
    h3,
    span {
      color: red;
    }

    p {
      color: blue;
      text-decoration: underline;
    }

    .myAbc {
      color: green;
      border: 1px solid grey;
      font-size: 20px;
    }

    p>i>b {
      font-size: 40px;
      color: cadetblue;
    }

    span>p:first-child {
      background-color: grey;
    }
```



```

a:hover {
    color: pink;
}

p:first-letter {
    font-size: 400%;
}

#book1 {
    background-color: cyan;
}

#book2 {
    background-color: aliceblue;
}
</style>

</head>

<body>

<div class="shadow">This box has a shadow</div>
<p>Lorem ipsum dolor sit amet consectetur.
</p>
<h1 class="myAbc">This demo for CSS</h1>
<h1 class="normal">Shri ramdeobaba college of Engg & management</h1>
<h1 class="tight">Shri ramdeobaba college of Engg & management</h1>
<h1 class="loose">Shri ramdeobaba college of Engg & management</h1>
<h1 class="upper">Shri ramdeobaba college of Engg & management</h1>
<h1 class="lower">Shri ramdeobaba college of Engg & management</h1>
<h1 class="normal">Shri ramdeobaba college of Engg & management</h1>

<h2>This is my text</h2>
<h3>this is my text</h3>
<span>what the span</span>
<span>what the span</span>
<div id="book1">what the div
    <h1>This is me</h1>
</div>
<div id="book2">what the div
    <h1>This is me</h1>
</div>

```

```

<p class="myAbc">This is a normal paragraph</p>
<P style="color:purple; font-size : 22px;background-color:yellow">Important : This is a
Notice for students</P>
<P>This is a fourth paragraph</P>
<p>This is a praragraph to <i>demonstrante <b>some blod text</b></i></p>
<span>
  <p>champion</p>
  <p>new</p>
</span>
<a href="https://googl.com">click here</a>
</body>
</html>

```

### Style.css

```

.normal {
  letter-spacing: normal;
}

.tight {
  letter-spacing: -.2em;
}

.loose {
  letter-spacing: .2em;
}

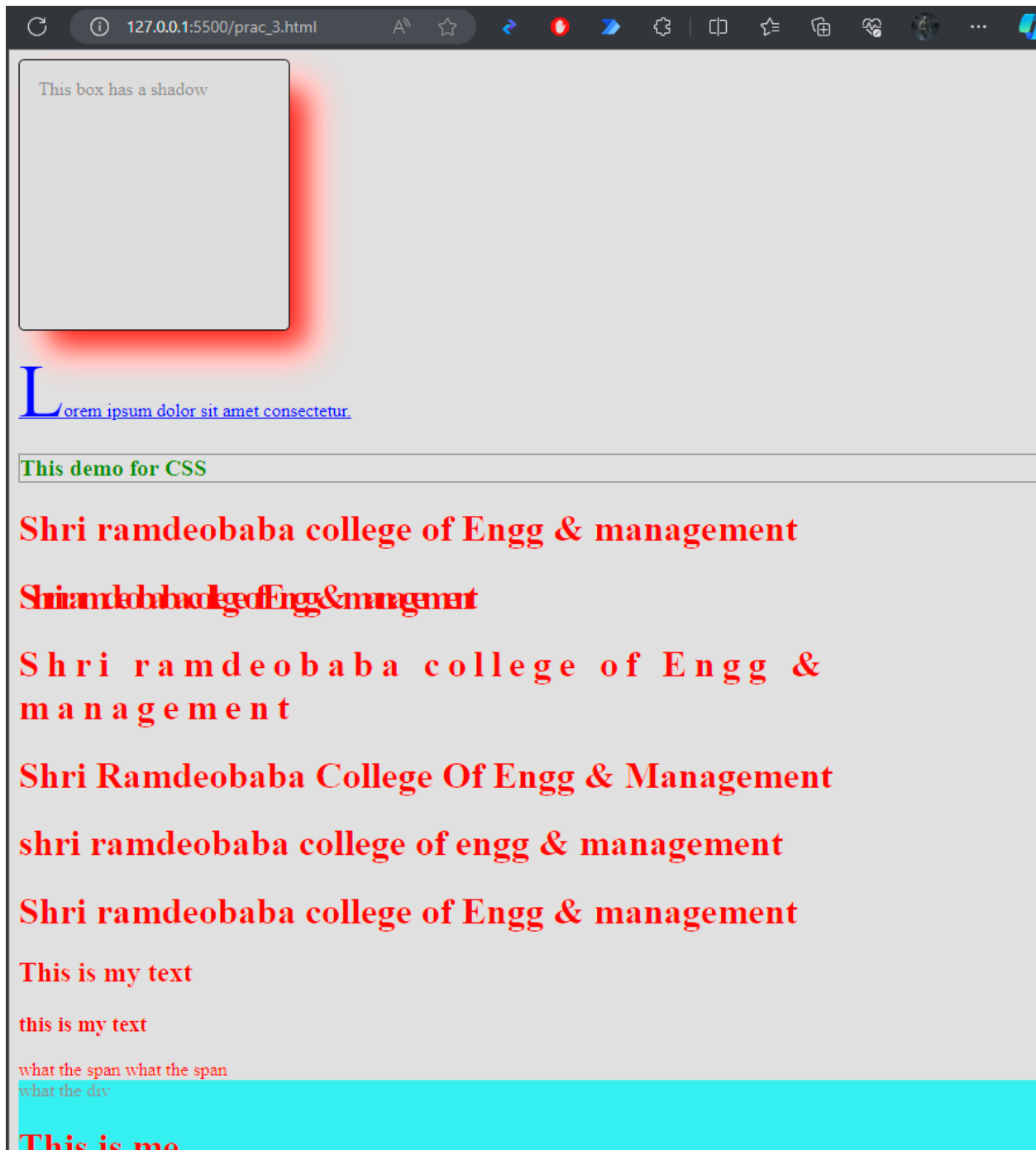
.upper {
  text-transform: capitalize;
}

.lower {
  text-transform: lowercase;
}

.shadow {
  box-shadow: 20px 20px 30px red;
  height: 200px;
  width: 200px;
  padding: 1em;
  border: 1px solid black;
  border-radius: 5px;
  background-color: #EEEEEE;
}

```

## Output:



## Practical 4

**Practical name:** Installation of NodeJs and setting up of server using NodeJs.

Steps to install nodejs and setting npm server on it

1. Download and Install Node.js:
  - a. Visit the official Node.js website at <https://nodejs.org/>.
  - b. Download the appropriate installer for your operating system (Windows, macOS, or Linux).
  - c. Run the installer and follow the prompts to complete the installation.
2. Verify Installation:
  - a. Open a command prompt or terminal window.
  - b. Type the following command to check if Node.js is successfully installed:

```
node -v npm -v
```

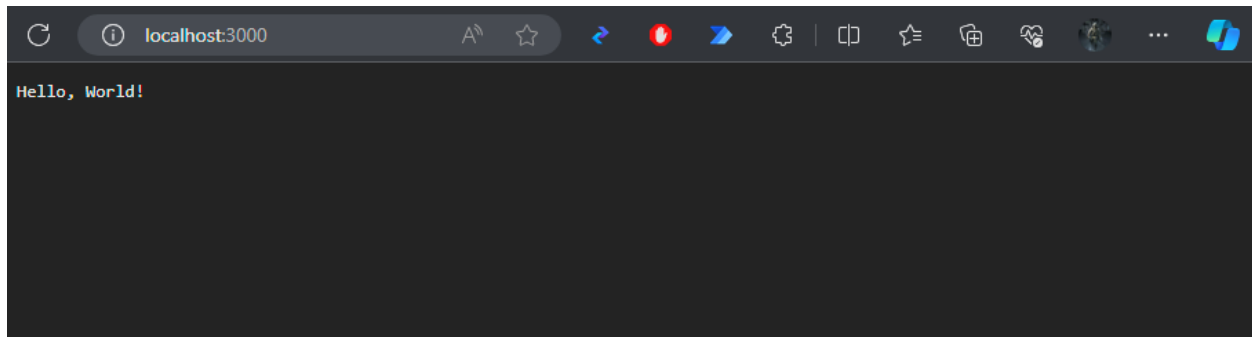
create a javascript file with this code:

```
const http = require('http');
const server = http.createServer((req, res) => {
  res.statusCode = 200; res.setHeader('Content-Type', 'text/plain');
  res.end('Hello, World!');}); const port = 3000; server.listen(port, () => {
  console.log(`Server running on port ${port}`);
});
```

this code will create a port on 3000 in which our server is running.

3. In terminal write: node filename.js

**Output:**



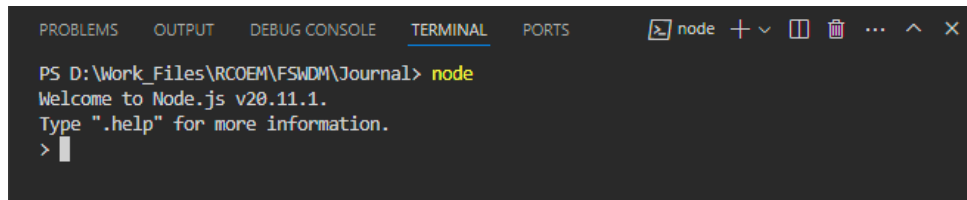
By, following this steps your server will be created.

## Practical 5

**Practical name:** Knowing NodeJs Repl environment and basic Commands.

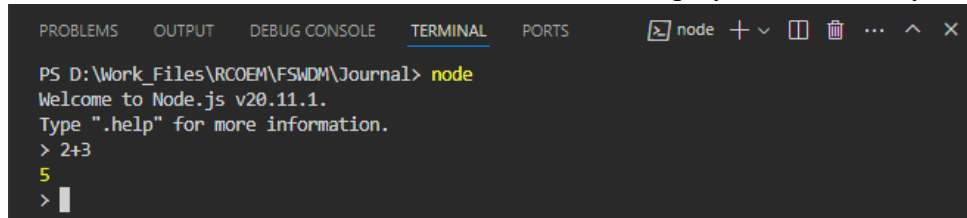
Node.js provides a REPL (Read-Eval-Print Loop) environment, which allows you to interactively run JavaScript code and see the results immediately.

1. Open a command prompt or terminal window.
  - a. Type node and press Enter.
  - b. You will see the Node.js REPL prompt (>), indicating that you are now in the REPL environment.:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS node + - [ ] [ ] ... ^ X
PS D:\Work_Files\RCOEM\FSWDM\Journal> node
Welcome to Node.js v20.11.1.
Type ".help" for more information.
> 
```

2. Run JavaScript Code:
  - a. Simply type JavaScript code at the REPL prompt and press Enter.
  - b. The code will be executed, and the result will be displayed immediately.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS node + - [ ] [ ] ... ^ X
PS D:\Work_Files\RCOEM\FSWDM\Journal> node
Welcome to Node.js v20.11.1.
Type ".help" for more information.
> 2+3
5
> 
```

You can use various commands and special keys in the Node.js REPL:

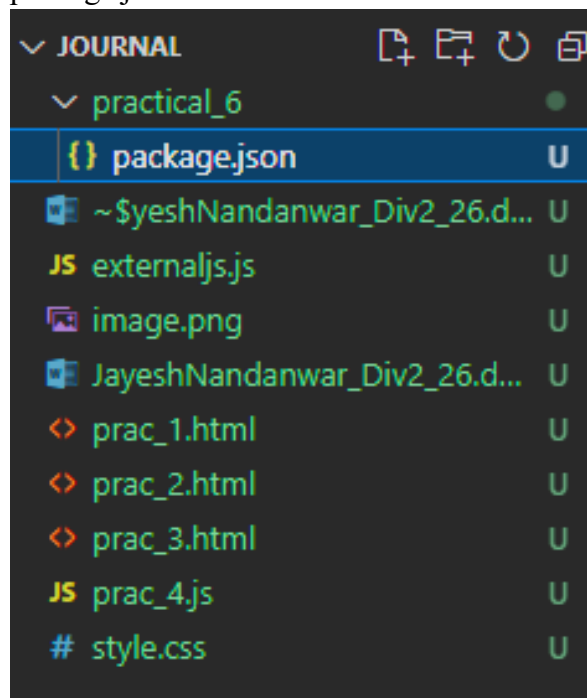
- .help or .h: Display the list of available commands.
- .break or .b: Exit from a multi-line expression.
- .clear or .c: Clear the REPL buffer.
- .exit or .ctrl + c: Exit the REPL.

The Node.js REPL is a useful tool for experimenting with JavaScript code, testing small snippets, and debugging. It allows you to quickly iterate and see the immediate results, making it a valuable environment for learning and prototyping.

## Practical 6

**Practical name:** Creating a Node Module and using NPM packages.

1. Initialize a Node.js project:
  - a. Create a new directory for your project.
  - b. Open a command prompt or terminal window.
  - c. Navigate to the project directory using the cd command.
  - d. Run the npm init command to initialize a new Node.js project and create a package.json file:



we, have created this package.json file inside the journal6 directory.

2. Create a Node Module:
  - a. Inside your project directory, create a new JavaScript file. This will be your Node module.
  - b. Write your module code in the JavaScript file.
3. Use the Node Module:
  - a. In another JavaScript file or in the Node.js REPL, you can use the module you created.
  - b. Require the module using the require() function and assign it to a variable.
4. Install and Use NPM Packages:

- a. NPM (Node Package Manager) allows you to easily install and use third- party packages in your Node.js projects.
- b. Search for the desired package on the NPM website use the npm search command to find packages from the command line.
- c. Install a package by running the npm install command followed by the package name.:

**npm install package-name**

5. Manage dependencies:

- a. When you install packages using npm install, the dependencies are automatically added to your package.json file.
- b. To install all the dependencies listed in package.json, run npm install without any package name.
- c. You can also specify package versions or ranges in the dependencies section of package.json to control which versions are installed.

## Practical 7

**Practical name:** Demonstrating the Asynchronous programming using node.js.

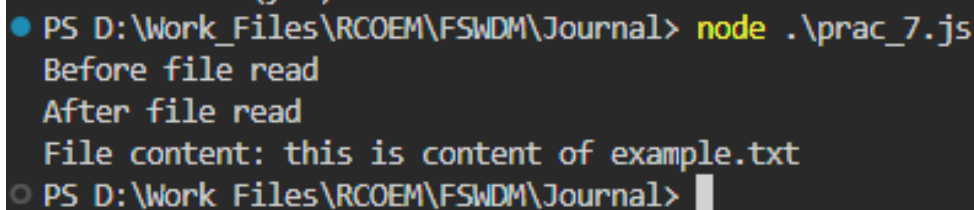
What is Asynchronous?

- The term Asynchronous means not occurring or happening at the same time. It is the opposite of synchronous, which means the events which are occurring at the same time.
- It defines the relationship between two or more objects that interact with each other within the same environment or system, but at different time phases or time instant, and also does not affect each other such as **Offline Learning**. In offline learning, the same content is available for all the candidates, but each candidate takes the sessions at any random time that does not match with other candidates.
- We can also understand it with a simple example of an email, as in this, people respond as per their convenience; hence it is an asynchronous event.

**Code for asynchronous programming:**

```
const fs = require('fs'); console.log('Before file read');
fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Error:', err);
  } else {
    console.log('File content:', data);
  }
});
console.log('After file read');
```

**Output:**



```
PS D:\Work_Files\RCOEM\FSWDM\Journal> node .\prac_7.js
Before file read
After file read
File content: this is content of example.txt
PS D:\Work_Files\RCOEM\FSWDM\Journal>
```



## Practical 8

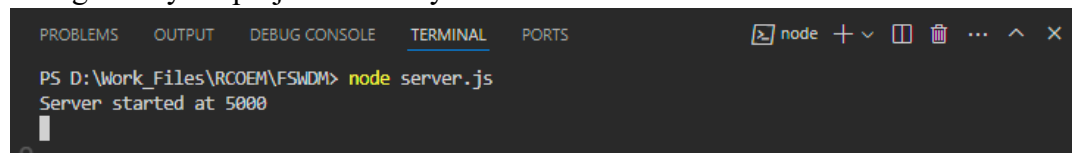
**Practical name:** Installation of Express and launching the server using Expressjs.

Step To install Express and launch a server using Express.js:

1. Create a new directory for your project.
2. Open a command prompt or terminal window.
3. Navigate to the project directory using the cd command.
4. Run the following command to initialize a new Node.js project and create a package.json file: **npm init**
5. Install Express:
  - a. Open a command prompt or terminal window.
  - b. Navigate to your project directory.
  - c. Use command: **npm install express**
6. Create a server file:
  - a. Inside your project directory, create a new JavaScript file. This will be your server file. Code for setting up the server in express:

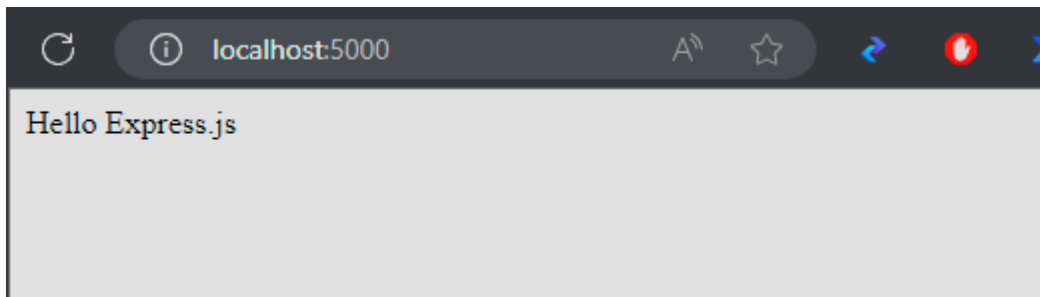
```
const express = require('express')
const app = express()
app.get('/', function (req, res) {
  res.send('Hello Express.js')
})
app.listen(5000)
console.log("Server started at 5000")
module.exports = app;
```

7. Launch the server:
  - a. Open a command prompt or terminal window.
  - b. Navigate to your project directory:



```
node server.js
Server started at 5000
```

**Output:**



## Practical 9

**Practical name:** Implementing MVC pattern for Express for Express.js Application

What is Model-View-Controller?

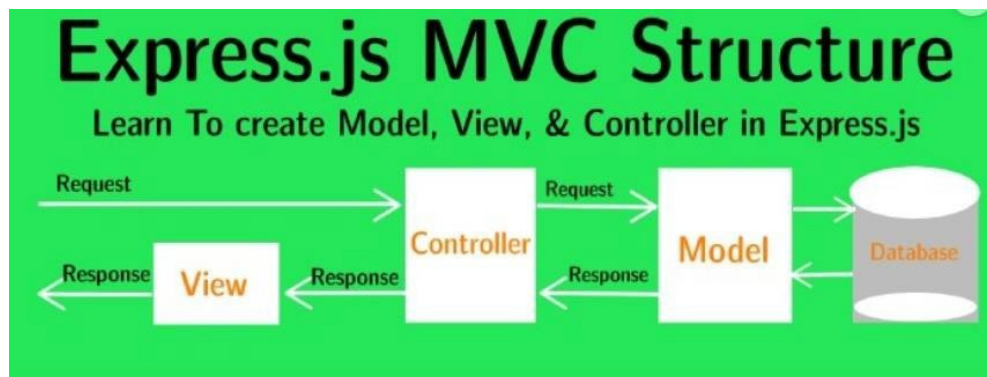
MVC is simply a design or architectural pattern used in software engineering. While this isn't a hard rule, but this pattern helps developers focus on a particular aspect of their application, one step at a time.

The main goal of MVC is to split large applications into specific sections that have their own individual purpose. It also allows developers to logically structure applications in a secure way, which we will show in this tutorial. But first, let's break down what each aspect of the pattern provides.

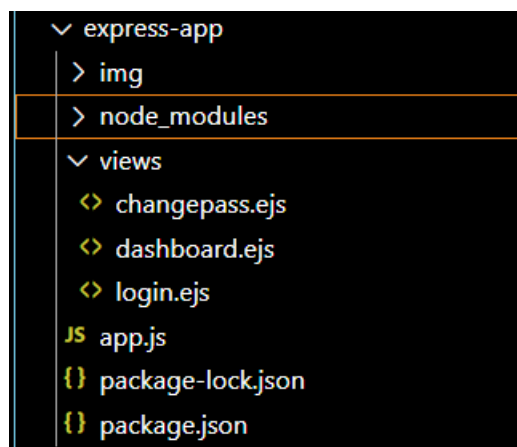
1. Model:
  - a. In the case of MVC, the model determines how a database is structured, defining a section of the application that interacts with the database. This is where we will define the properties of a user that will be stored in our database.
  - b. The controller accesses the database through the model. You could say that the model is the heart of the application.
2. View:
  - a. The view is where end users interact within the application. Simply put, this is where all the HTML template files go.
3. Controller:
  - a. The controller interacts with the model and serves the response and functionality to the view. When an end user makes a request, it's sent to the controller which interacts with the database.
  - b. You can think of the controller as a waiter in a restaurant that handles customers' orders, which in this case is the view. The waiter then goes to the kitchen, which is the model/database, and gets food to serve the customers, which is the controller handling the request.

Requirements to create a MVC model:-

- Node.js
- A MongoDB Atlas account to create our app's database
- Yarn if you are not using npm with Node.js
- A text editor, such as Visual Studio Code



First of all, Install Express app: **npm install express**



**Model** - In this folder, you can write the functionality & logics related to the Database like insert, fetch, update, delete queries. Even It takes the query request from the controller & sends the response back to the controller.

e.g code: `//models`

```
const users = [
  {
    id: 1, username: 'Jayesh', password: 'rcoem123', dob: '2002-10-19', email:
    jayesh@example.com'
  },
  // more user can be added
];
const User = {
  getAllUsers: () => {
    return users;
  },
  getUserByUsername: (username) => {
```

```

    return users.find((user) => user.username === username);
  },
  // Other model methods for creating, updating, deleting users...
};
module.exports = User;

```

**View** - In this folder, you can write HTML code for displaying a web page on the web browser. Even you can send the data from the controller to view for displaying data dynamically.

e.g code:

```

<!DOCTYPE html>
<html>
<head>
  <title>Login</title>
</head>
<body>
  <h1>Login</h1>
  <form action="/login" method="POST">
    <input type="text" name="username" placeholder="Username" required>
    <input type="password" name="password" placeholder="Password" required>
    <button type="submit">Login</button>
  </form>
</body>
</html>

```

**Controller** - In this folder, you can write the functionality & logic to develop dynamic web applications. Even it takes the data request from the views & sends it to the model and sends the response back to the views.

e.g code-:

```

const User = require('../models/user');
const userController = {
  login: (req, res) => {
    res.render('login');
  },
  authenticate: (req, res, next) => {
    const { username, password } = req.body;
    const correctUsername = 'yash'; const correctPassword = 'rcoem123';
    if (username === correctUsername && password === correctPassword) {
      next();
    } else {

```

```

        res.redirect('/');
    }
},
performLogin: (req, res) => {
    const { username, dob, email } = req.body;
    req.session.user = {
        username,
        dob, email
    };
    res.redirect('/dashboard');
},
dashboard: (req, res) => {
    const { user } = req.session;
    if (!user) {
        res.redirect('/');
    } else {
        res.render('dashboard', { user });
    }
},
changePassword: (req, res) => {
    res.render('changepass');
},
performChangePassword: (req, res) => {
    const { newPassword } = req.body;
    res.redirect('/dashboard');
}
};
module.exports = userController;

```

## Practical 10

**Practical name:** Rendering of views and static files using Express.js.

1. To Set up the View Engine:
  - a. Install the EJS view engine package using npm:  
**npm install ejs**
2. Render the View:
  - a. Set up a route in your Express app to handle a specific URL.

**Code:**

```
app.get('/students',function(req,res){
  res.render('index.ejs',{
    title:"RCOEM MCA Students",
    users:students
  })
})
```

In this route handler, we render the students view template and pass a variable users with the values to the view.

### **Npm install express**

App.js file to render the files on express

```
var express = require('express');
var app = express();

var bodyparser = require('body-parser');
var path = require('path');

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

var students=[
  {
    id:1,
    first_name:"abc",
    last_name:"xyz",
    email:"abc@xyz.ccc"
  },
```

```

    {
      id:2,
      first_name:"abc11",
      last_name:"xyz22",
      email:"abc11@xyz22.ccc"
    },
    {
      id:3,
      first_name:"123abc",
      last_name:"123xyz",
      email:"123abc@123xyz.ccc"
    }
  ];

  app.get('/students',function(req,res){
    res.render('index.ejs',{
      title:"RCOEM MCA Students",
      users:students
    })
  })

  app.get('/login',function(req,res){
    res.sendFile(__dirname+'/loginPage.html')
  })

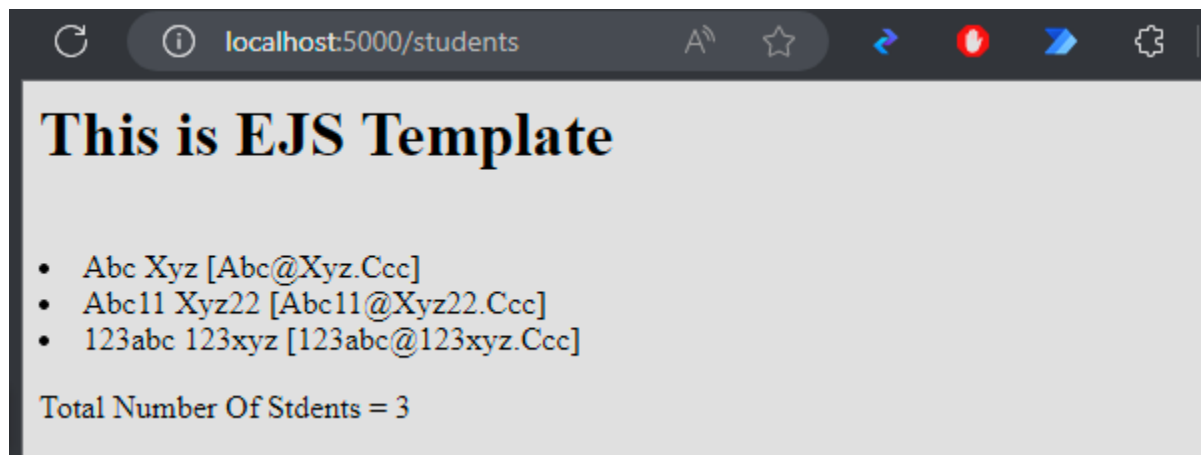
  app.use(bodyParser.urlencoded({extended:true}))
  app.post('/login',function (req,res) {
    var username =req.body.username;
    var password = req.body.password;
    if (username=='admin' && password == 'admin') {
      res.redirect('/students')
      // res.send("login Successful")
    }else{
      res.send('Wrong Credentials')
    }
  })

  var port=5000
  app.listen(port,function(){
    console.log("Server started at port:"+port)
  })

```



Output:



## Practical 11

**Practical name:** Implementing of web routes using express js.

1. Create an Express Application:

- a. Set up your Express application by requiring the express module and creating an instance of the express application.
- b. Javascript:

```
1  const express = require('express');
2  const session = require('express-session');
3  const cookieParser = require('cookie-parser');
4  const bodyParser = require('body-parser');
5
6  // Express application created
7  const app = express();
```

2. Define Routes:

- a. Use the app.get() method to define a route for handling GET requests. You can also use other HTTP methods like app.post(), app.put(), app.delete() for handling other types of requests.
- b. Specify the URL path for the route as the first argument and provide a callback function as the second argument. The callback function takes the request (req) and response (res) objects as parameters and handles the logic for that route.

```
app.get('/login',function(req,res){
    res.sendFile(__dirname+'/loginPage.html')
})
```

3. Handling POST Requests:

- a. To handle POST requests, use the app.post() method and provide a callback function similar to the app.get() method.
- b. Example route handling a POST request:

```
42  app.post('/login', authenticate, (req, res) => {
43      const { username, dob, email } = req.body;
44
45      // user ka data save krke dashboard print karane ke liye
46      req.session.user = {
47          username,
48          dob,
49          email
50      };
51      // redirect to dashboard
52      res.redirect('/dashboard');
53  });
```

4. Start the Server:

- a. Add the following code at the end of your script to start the Express server:

```
75 // we can give any port i have choosed 8080
76 const port = 8080;
77 app.listen(port, () => {
78   // the {port} will take the value of port inn which server is running
79   console.log(`Server running on http://localhost:\${port}`);
80 });
```

The combined code in which web routes using expressjs are implemented.

**Code:**

```
const express = require('express');
const session = require('express-session'); const cookieParser = require('cookie-parser'); const
bodyParser = require('body-parser');

const app = express()
app.use(cookieParser()); app.use(bodyParser.urlencoded({ extended: true })); app.use(session({
  secret: 'secret-key', resave: true, saveUninitialized: true
}));
app.set('view engine', 'ejs');

const authenticate = (req, res) => {
  const correctUsername = 'jayesh';
  const correctPassword = 'rcoem123';

  const { username, password } = req.body;

  if (username === correctUsername && password === correctPassword) {
    res.send("Login Successful");
  } else {
    res.send("Wrong Credentials");
  }
};

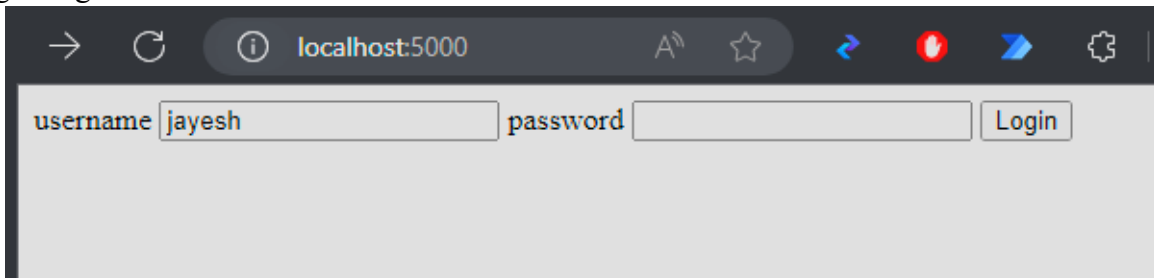
app.get('/', (req, res) => {
  res.render('login');
});

app.post('/login', authenticate, (req, res) => {
  const { username, dob, email } = req.body;
  console.log(username, dob, email)
```

```
req.session.user = {  
  username, dob, email  
};  
});  
  
app.listen(5000)  
console.log("Server started at 5000")
```

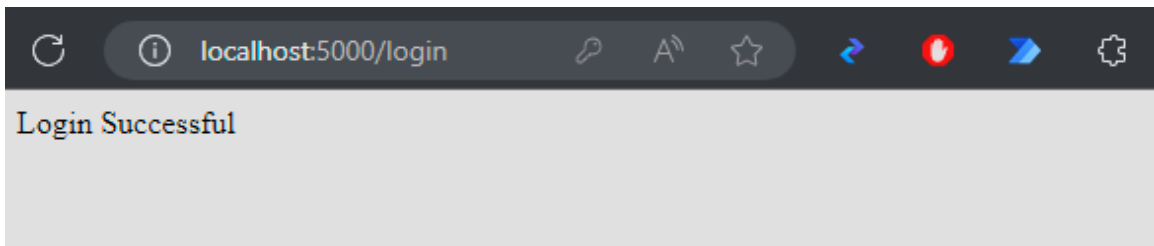
### Output:

Login Page

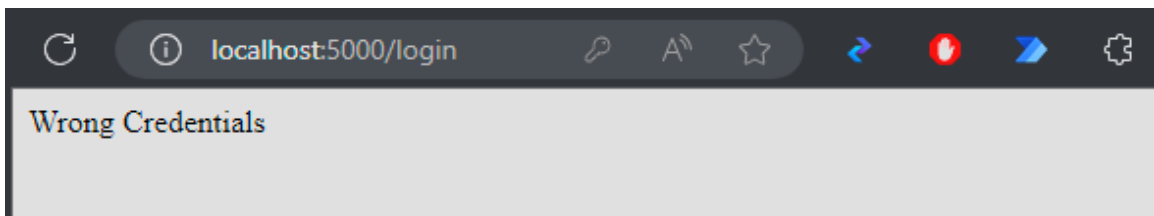


username  password

Correct Input:



Wrong Input:



## Practical 12

**Practical Name:** Using the cookies and session for persisting the user data.

### Cookies:

- Cookies are small pieces of data stored on the client's browser.
- They are sent by the server to the client's browser and are included with subsequent requests to the same server.
- Cookies are commonly used for tracking user authentication, session state, preferences, and other user-specific information.
- They have a limited size (usually up to 4KB) and can store simple key-value pairs.

### Sessions:

- Sessions are server-side data storage mechanisms used to maintain stateful information about a user's interactions with a website or web application.
- Unlike cookies, which are stored on the client's browser, session data is stored on the server.
- Sessions are typically implemented using unique session identifiers (usually stored in cookies) that allow the server to associate subsequent requests from the same client with the same session data.
- Session data can store more complex information compared to cookies and is not limited by size constraints.
- Sessions are commonly used for user authentication, user-specific data storage, and maintaining temporary state throughout a user's interaction with a web application.

### Code:

```
// app.js

const express = require('express');
const cookieParser = require('cookie-parser');
const session = require('express-session');

const app = express();
const PORT = 3000;

// Middleware
app.use(cookieParser());
app.use(session({
  secret: 'secret', // Change this to a random string
  resave: false,
```

```

    saveUninitialized: true
  }));

// Routes
app.get('/', (req, res) => {
  let username;
  // Check if the user has a session
  if (req.session.username) {
    username = req.session.username;
  } else {
    // Check if username exists in cookies
    username = req.cookies.username;
  }

  if (username) {
    res.send(`Welcome back, ${username}!`);
  } else {
    res.send('Welcome! Please log in.');
```

```

  });

  app.get('/login', (req, res) => {
    const username = 'user123'; // Hardcoded for demonstration purposes, in real world scenario
    you would validate the credentials
    req.session.username = username;
    res.cookie('username', username, { maxAge: 900000, httpOnly: true }); // Set cookie to expire
    in 15 minutes
    res.redirect('/');
  });
```

```

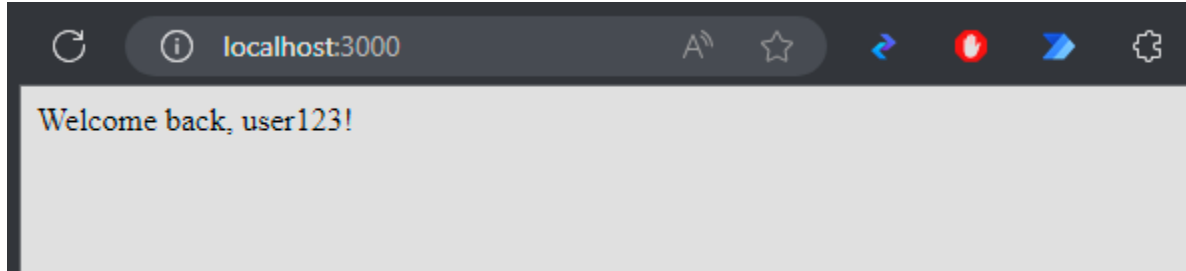
  app.get('/logout', (req, res) => {
    req.session.destroy(err => {
      if (err) {
        console.error('Error destroying session:', err);
        res.status(500).send('Error logging out');
      } else {
        res.clearCookie('username');
        res.redirect('/');
      }
    });
  });
```

```

  // Start the server
```

```
app.listen(PORT, () => {  
  console.log(`Server is listening on port ${PORT}`);  
});
```

**Output:**



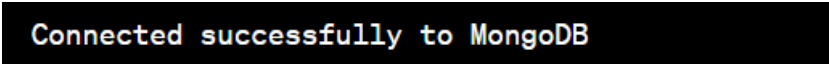
## Practical 13

**Practical name:** Installing MongoDB instance and to run it for accessing through client.

**Code:**

```
const { MongoClient } = require('mongodb');
const url = 'mongodb://localhost:27017';
const dbName = 'my_database';
const client = new MongoClient(url, { useUnifiedTopology: true });
client.connect((err) => {
  if (err) {
    console.error('Error connecting to MongoDB:', err);
    return;
  }
  console.log('Connected successfully to MongoDB');
  const db = client.db(dbName);
  client.close();
});
```

**Output:**

A black rectangular box with the text "Connected successfully to MongoDB" in a light blue, monospaced font, representing a terminal output.

```
Connected successfully to MongoDB
```



## Practical 14

**Practical name:** Creation of databases and collections using MongoDB.

```
PS C:\Users\LAPTOP HOUSE> mongod
{"t":{"$date":"2023-07-20T01:44:27.789+05:30"},"s":"I", "c":"CONTRO
sabbling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtoc
{"t":{"$date":"2023-07-20T01:44:27.792+05:30"},"s":"I", "c":"NETWOR
specification","attr":{"spec":{"incomingExternalClient":{"minWireVe
t":{"minWireVersion":0,"maxWireVersion":13},"outgoing":{"minWireVers
2}
```

```
PS C:\Users\LAPTOP HOUSE> mongo
MongoDB shell version v5.0.3
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("3f36f0bb-6a8c-4034-94fb-2cbea4297d1e") }
MongoDB server version: 5.0.3
=====
```

```

---
> db
test
> show dbs
admin          0.000GB
config         0.000GB
harrykart     0.000GB
local         0.000GB
movies        0.000GB
> use harrykart
switched to db harrykart
> use harshal
switched to db harshal
> db.createCollection("mycollection")
{ "ok" : 1 }
>

```

## Practical 15

**Practical name:** Implementing the CRUD operations using MongoDB queries.

**Create Operation:**

```
Windows PowerShell
> show dbs
admin      0.000GB
config     0.000GB
harrykart  0.000GB
harshal    0.000GB
local      0.000GB
movies     0.000GB
> use harshal
switched to db harshal
> db.harshal.insertOne({name:"Champion"})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("64b8d472a1a3dbd0bda69af4")
}
> |
```

```
> show dbs
admin      0.000GB
config     0.000GB
harrykart  0.000GB
harshal    0.000GB
local      0.000GB
movies     0.000GB
> use harshal
switched to db harshal
> db.harshal.insertMany([ {name:"elliott"}, {name:"darlene"} ])
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("64b8d52ca1a3dbd0bda69af5"),
    ObjectId("64b8d52ca1a3dbd0bda69af6")
  ]
}
> |
```

## Read Operations:

```
mote systems will be unable to connect to this server. Start the server with
> show dbs
admin      0.000GB
config     0.000GB
harrykart  0.000GB
harshal    0.000GB
local      0.000GB
movies     0.000GB
> use harshal
switched to db harshal
> db.harshal.find()
{ "_id" : ObjectId("64b8d472a1a3dbd0bda69af4"), "name" : "Champion" }
{ "_id" : ObjectId("64b8d52ca1a3dbd0bda69af5"), "name" : "elliot" }
{ db.harshal.find({name:"elliot"})d0bda69af6", "name" : "darlene" }
{ "_id" : ObjectId("64b8d52ca1a3dbd0bda69af5"), "name" : "elliot" }
> |

> use harshal
switched to db harshal
> db.harshal.findOne({name:"elliot"})
{ "_id" : ObjectId("64b8d52ca1a3dbd0bda69af5"), "name" : "elliot" }
> |
```

## Update Operations:

```
2023-07-20T11:58:58.387+05:30: This server is bound to localhost. Re> use harshal
switched to db harshal
> db.harshal.updateOne({name:"Champion"}, {$set: {name : "Harshal"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> |
```

### Delete Operations:

```
> use harshal
switched to db harshal
> db.harshal.deleteOne({name:"elliott"})
{ "acknowledged" : true, "deletedCount" : 1 }
> |
```

## Practical 16

**Practical name:** Connecting Express.Js webpages through Mongoose for MongoDB.

```
npm install mongoose App.js
```

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

```
const app = express();
```

```
const mongoose = require('mongoose'); const bodyparser = require("body-parser");
```

```
mongoose.connect('mongodb://localhost/contactDance', { useNewUrlParser: true });
```

```
const port = 8000;
```

```
const contactSchema = new mongoose.Schema({
```

```
  name: String,
```

```
  phone: String, email: String, address: String, desc: String
```

```
});
```

```
const Contact = mongoose.model('Contact', contactSchema);
```

```
app.use('/static', express.static('static'))
```

```
app.set('view engine', 'pug')
```

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

```
  const params = {}
```

```
  res.status(200).render('home.pug', params);
```

```
})
```

```
app.get('/contact', (req, res) => {
```

```
  const params = {}
```

```
  res.status(200).render('contact.pug', params);
```

```
})
```

```
app.post('/contact', (req, res) => {
```

```
  var myData = new Contact(req.body); myData.save().then(() => {
```

```
    res.send("This item has been saved to the database")
```

```
  }).catch(() => {
```

```
    res.status(400).send("Item was not saved to the database")
```

```
  });
```

```
})
```

```
app.listen(port, () => {
```

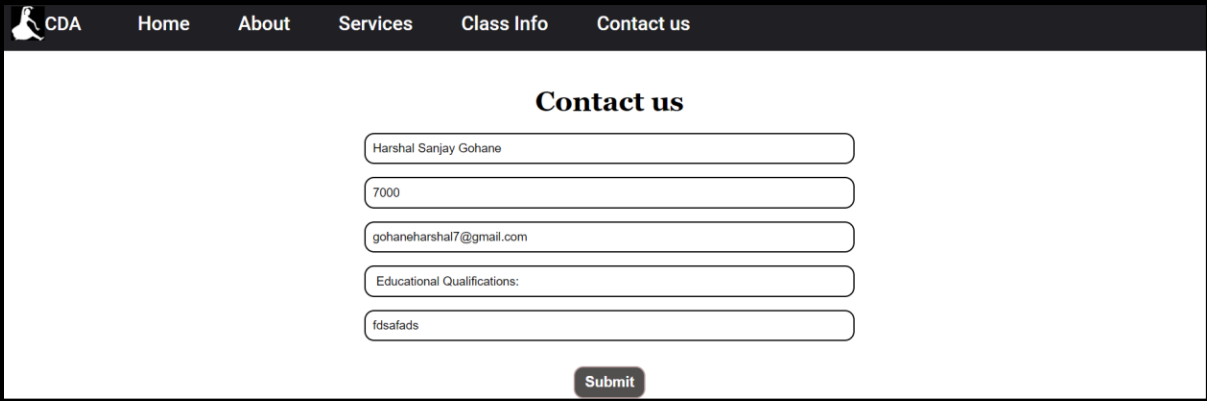
```
  console.log(`The application started successfully on port ${port}`);
```

```
});
```

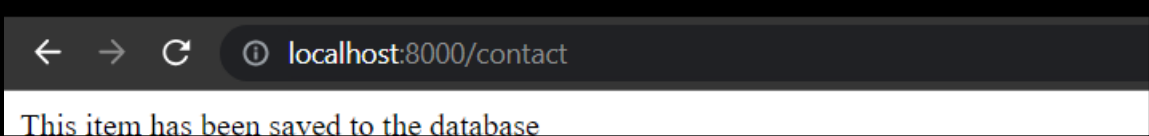
Views / base.pug

```
//- layout.pug doctype html html
head
title Harsh Dance Academy block scripts
block style
body nav#navbar
ul
div#logo img(src = "/static/1.png")
  | CDA
li #[a(href = "/") Home] li #[a(href = "/") About]
li #[a(href = "/") Services] li #[a(href = "/") Class Info]
li #[a(href = "/contact") Contact us]
block content
```

Output:



The screenshot shows a web browser window displaying a contact form. The browser's address bar shows the URL `localhost:8000/contact`. The page has a dark header with the CDA logo and navigation links: Home, About, Services, Class Info, and Contact us. The main content area is titled "Contact us" and contains a form with five input fields. The first four fields contain the text "Harshal Sanjay Gohane", "7000", "gohaneharshal7@gmail.com", and "Educational Qualifications:" respectively. The fifth field contains "fidsafads". A "Submit" button is located at the bottom right of the form.



The screenshot shows the bottom of the browser window. The address bar displays `localhost:8000/contact`. Below the address bar, a message bar shows the text "This item has been saved to the database".

## Practical 17

**Practical name:** Creating AngularJs modules for using in front-end web page.

App.module.ts

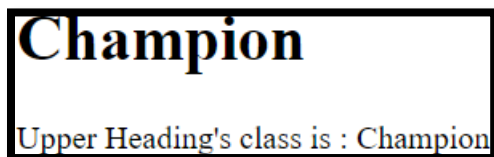
```
import { NgModule } from '@angular/core';
// BrowserModule automatically imports all CommonModule Dependencies

import { BrowserModule } from '@angular/platform-browser'; import { AppComponent } from
'./app.component';

@NgModule({
  declarations: [AppComponent
  ],
  imports: [
    // Adding Imports BrowserModule,
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { } app.component.ts
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  geek = "Champion";
  g = document.getElementsByClassName(this.geek);
}
App.component.html
< h1[ngClass] = "geek" > Champion</h1 >
Upper Heading's class is : "{{ g[0].className }}"
```

Output:



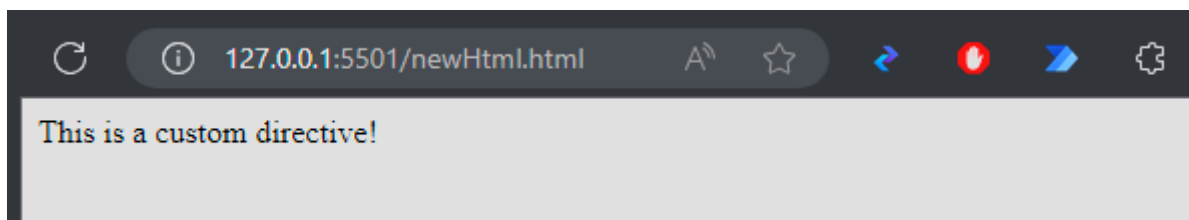
## Practical 18

**Practical name:** Demonstrate AngularJS modules and Directive in an HTML page.

```
<!DOCTYPE html>
<html lang="en" ng-app="myApp">

<head>
  <meta charset="UTF-8">
  <title>AngularJS Modules and Directives</title>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.8.2/angular.min.js">
  </script>
</head>
<body>
  <my-directive></my-directive>
  <script>
    var app = angular.module('myApp', []);
    app.directive('myDirective', function () {
      return {
        restrict: 'E',
        template: '<div>This is a custom directive!</div>'
      };
    });
  </script>
</body>
</html>
```

Output:





## Practical 19

**Practical name:** Implementing Dependency Injection using AngularJs for a given webpage.

App.js

```
var app = angular.module('myApp', []);
app.service('myService', function () {
  this.getMessage = function () {
    return 'Hello from the service!';
  };
});
app.controller('MyController', ['$scope', 'myService', function ($scope, myService) {
  $scope.message = myService.getMessage();
}]);
```

index.html

```
<!DOCTYPE html>
<html lang="en" ng-app="myApp">
<head>
  <meta charset="UTF-8">
  <title>AngularJS Dependency Injection</title>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.8.2/angular.min.js">
  </script>
</head>
<body>
  <div ng-controller="MyController">
    {{ message }}
  </div>
  <script src="app.js"></script>
</body>
</html>
```

Output:

