| **ExNo.** | **1** | **Create a form and validate the contents of the form using JavaScript** |
| --- | --- | --- |
| **DATE** |  |

# Aim:

# To create a form and validate the form using JavaScript.

**Algorithm:**

**Step 1:** start the visual studio code.

**Step 2**: As this is a full stack development we are going to front end and the back end.

**Step 3:** We will be using html, CSS for style and JavaScript.

**Step 4:**In validation form we are using login and password.

**Step 5:**Here, we are validating the form on form submit. The user will not be forwarded to the next page until given values are correct.

**Step 6:** After using the correct login and password the user is allowed to move to the next page.

**Step 7:** Thus the created form is validated the using JavaScript.

**Program:**

#index.html

<html>

  <head>

    <meta name="viewport" content="width=device-width, initial-scale=1">

    <link rel="stylesheet" href="/css/styles.css">

    <title>LOGIN PAGE</title>

<body>

    <h1>Login</h1>

    <form name="login" action="#">

      <div class="row">

        <label for="username">Username</label>

        <input type="text" name="username" />

      </div>

      <div class="row">

        <label for="password">Password</label>

        <input type="password" name="password" />

      </div>

      <input type="button" class="button" value="Login" onclick="Login()" />

    </form>

    <script  type="text/javascript">

        function Login() {

            var done = 0;

            var username = document.login.username.value;

            username = username.toLowerCase();

            var password = document.login.password.value;

            password = password.toLowerCase();

            if (username == "user" && password == "user@123") {

              alert("login success");

            //   window.open("https://google.com");

            } else if (done == 0) {

              alert("check for valid username and password");

           }

         }

    </script>

    <style>

@import url('https://fonts.googleapis.com/css?family=Open+Sans&display=swap');

  body {

    font-family: 'Open Sans', sans-serif;

    background: #f9faff;

    color: #3a3c47;

    line-height: 1.6;

    display: flex;

    flex-direction: column;

    align-items: center;

    margin: 0;

    padding: 0;

  }

  h1 {

    margin-top: 48px;

  }

  form {

    background: #fff;

    max-width: 360px;

    width: 100%;

    padding: 58px 44px;

    border: 1px solid #e1e2f0;

    border-radius: 4px;

    box-shadow: 0 0 5px 0 rgba(42, 45, 48, 0.12);

    transition: all 0.3s ease;

  }

  .row {

    display: flex;

    flex-direction: column;

    margin-bottom: 20px;

  }

  .row label {

    font-size: 13px;

    color: #8086a9;

  }

  .row input {

    flex: 1;

    padding: 13px;

    border: 1px solid #d6d8e6;

    border-radius: 4px;

    font-size: 16px;

    transition: all 0.2s ease-out;

  }

  .row input:focus {

    outline: none;

    box-shadow: inset 2px 2px 5px 0 rgba(42, 45, 48, 0.12);

  }

  .row input::placeholder {

    color: #C8CDDF;

  }

  .button {

    width: 100%;

    padding: 12px;

    font-size: 18px;

    background: #15C39A;

    color: #fff;

    border: none;

    border-radius: 100px;

    cursor: pointer;

    font-family: 'Open Sans', sans-serif;

    margin-top: 15px;

    transition: background 0.2s ease-out;

  }

  .button:hover {

    background: #55D3AC;

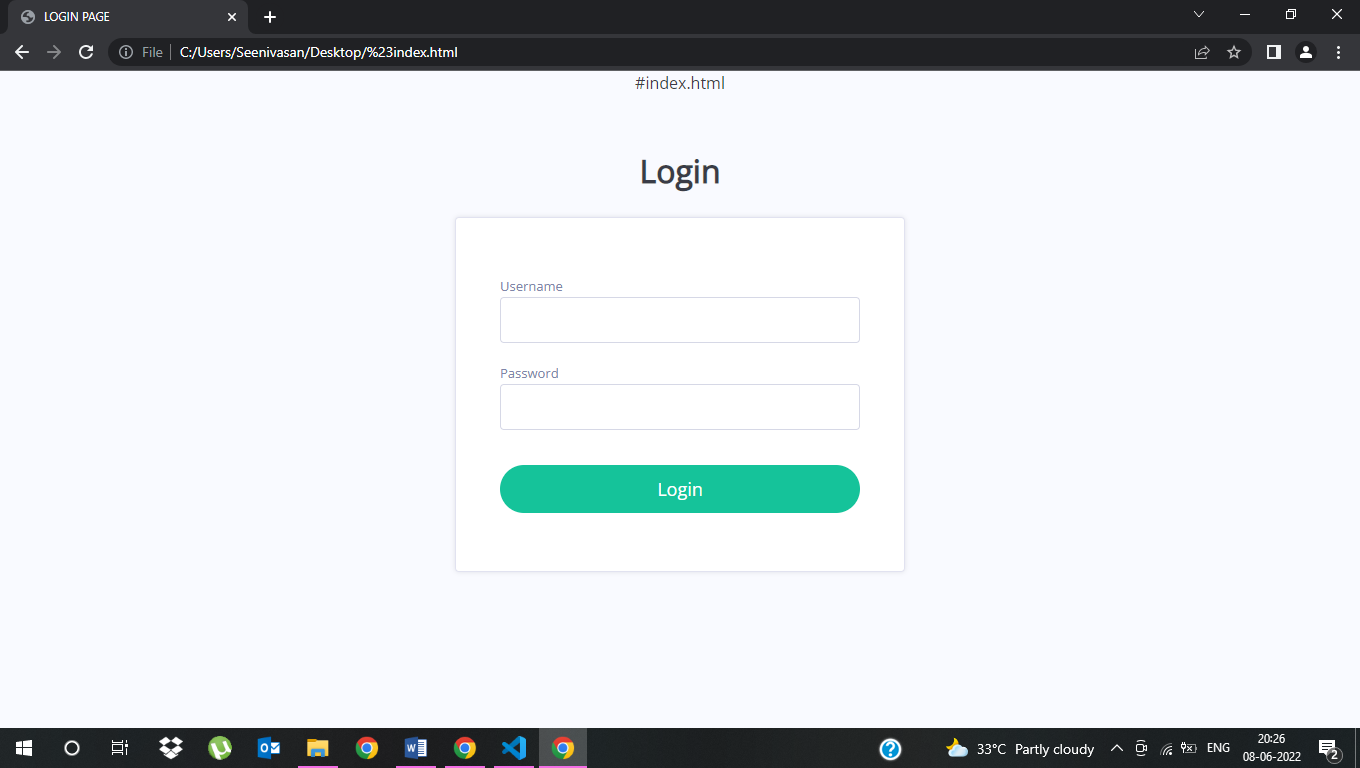
  }

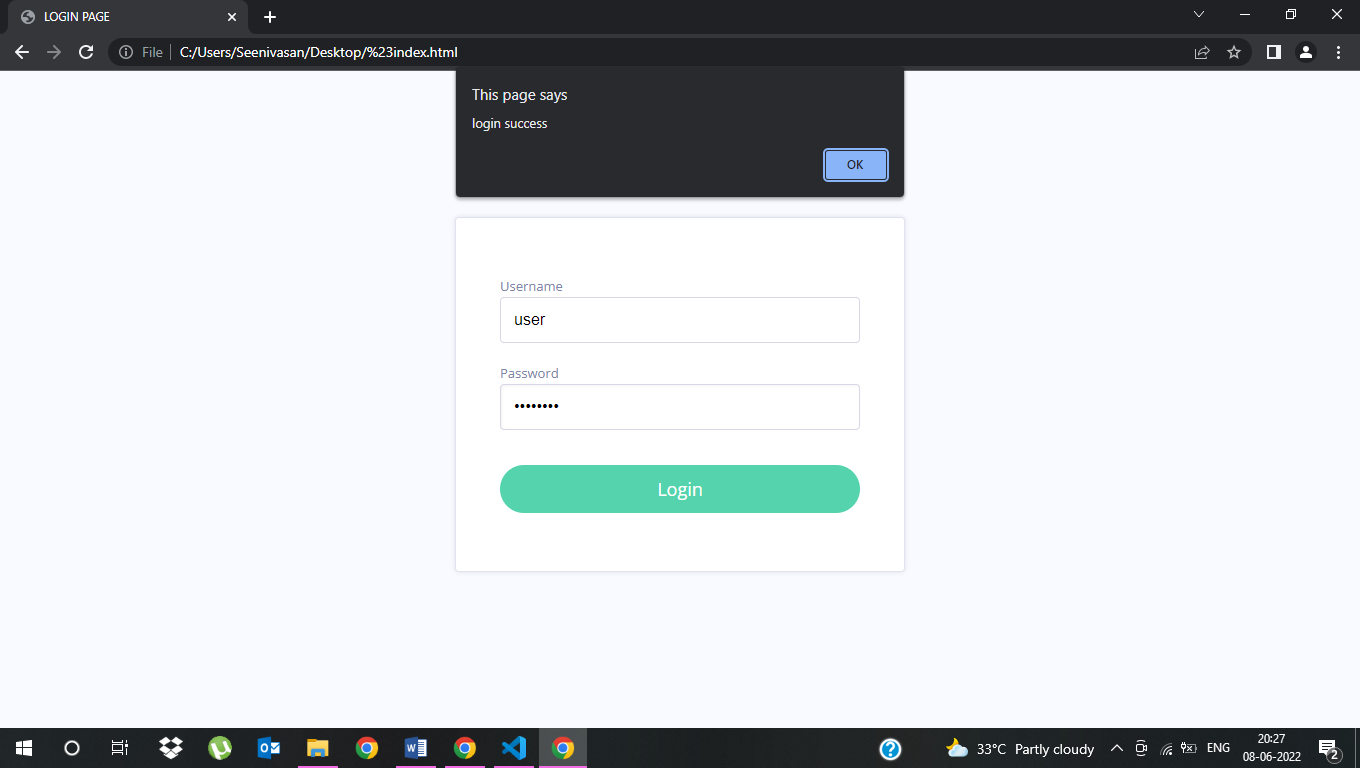
  </style>

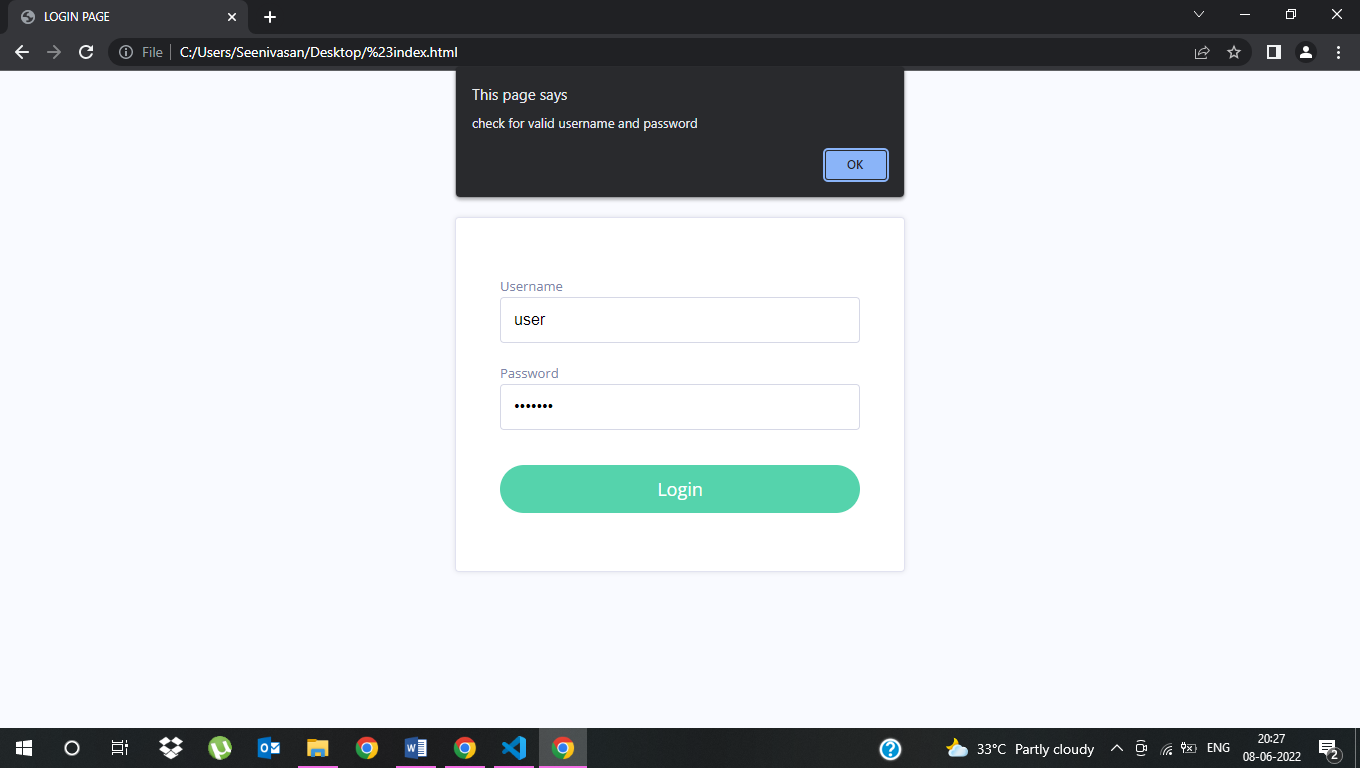
  </body>

</html>

**Output:**







**Result**

Thus, to create a form and validate the form using JavaScript was successfully completed.

| **ExNo.** | **2** | **Get data using Fetch API from an open source endpoint and display the contents in the form of a card** |
| --- | --- | --- |
| **DATE** |  |

# Aim:

# To Get data using Fetch API from an open source endpoint and display the contents in the form of a card.

**Algorithm:**

**Step 1**: Start the html coding.

**Step 2:**Using the fetch() method, fetch the API from open source.Using the JSON format , print the API details in the form of a card.Using the function displayCards(), with the help of map() method display the API details in the form of a card.

**Step 3:**fetch() method: The fetch() method is modern and versatile and is very well supported among the modern browsers. It can send network requests to the server and load new information whenever it’s needed, without reloading the browser.

**Step 4:**Save the coding with the help of .html extension. Open any one of the webbrowser, for viewing the output.

Syntax:

async function funcName(url){

const response = await fetch(url);

var data = await response.json();

}

**Step 5:**The **fetch() method** is used to send the requests to the server without refreshing the page. It is an alternative to the XMLHttpRequest object.  
The basic syntax of a fetch() request is as follows:

fetch(url, {options})

.then(data => {

// Do some stuff here

})

.catch(err => {

// Catch and display errors

})

The difference between XMLHttpRequest and fetch is that fetch uses [Promises](https://www.geeksforgeeks.org/javascript-promises/) which are easy to manage when dealing with multiple asynchronous operations where callbacks can create callback hell leading to unmanageable code.

**Step 6:**Stop the program.

**Program:**

#datafromapi.html

<!DOCTYPE html>

<html lang="en">

    <head>

        <script src="script.js"></script>

        <link rel="stylesheet" href="style.css" />

        <meta charset="UTF-8" />

        <meta name="viewport"

              content="width=device-width, initial-scale=1.0" />

        <title>Document</title>

    </head>

    <body>

        <!-- Here a loader is created which

             loads till response comes -->

        <div class="d-flex justify-content-center">

            <div class="spinner-border"

                 role="status" id="loading">

                <span class="sr-only">Loading...</span>

            </div>

        </div>

        <h1>data from Api</h1>

        <!-- table for showing data -->

        <table id="employees"></table>

        <script lang="javascript">

        const api\_url =

"https://api.sampleapis.com/coffee/hot";

// Defining async function

async function getapi(url) {

// Storing response

const response = await fetch(url);

// Storing data in form of JSON

var data = await response.json();

console.log(data);

if (response) {

hideloader();

}

show(data);

}

// Calling that async function

getapi(api\_url);

// Function to hide the loader

function hideloader() {

document.getElementById('loading').style.display = 'none';

}

// Function to define innerHTML for HTML table

function show(data) {

let tab =

`<tr>

<th>title</th>

<th>description</th>

<th>ingredients</th>

<th>image</th>

</tr>`;

// Loop to access all rows

for (let r of data) {

tab += `<tr>

<td>${r.title} </td>

<td>${r.description}</td>

<td>${r.ingredients}</td>

<td><img src=${r.image} alt="image not found"></img></td>

</tr>`;

}

// Setting innerHTML as tab variable

document.getElementById("employees").innerHTML = tab;

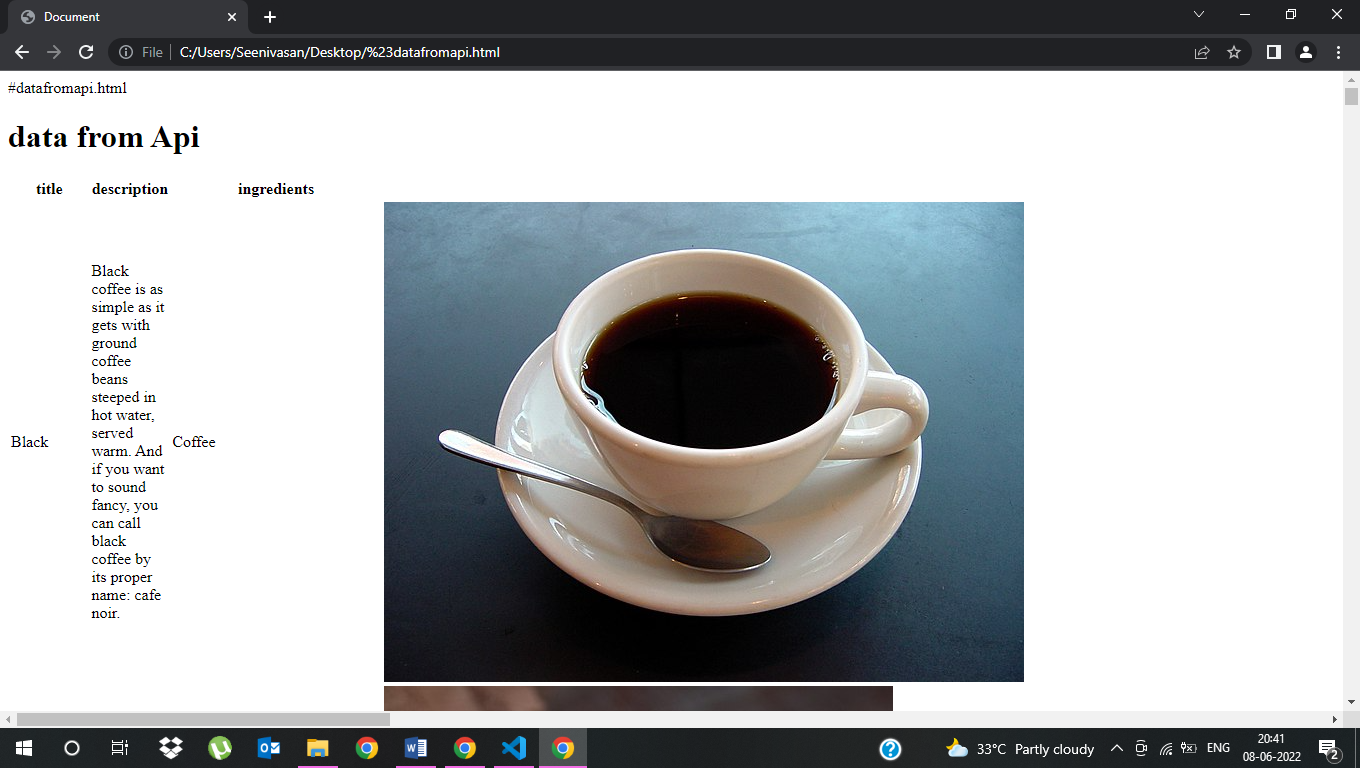
}

</script>

    </body>

</html>

**Output:**



# Result

# Thus, to Get data using Fetch API from an open source endpoint and display the contents in the form of a card was successfully completed.

| **ExNo.** | **3** | **Create a NodeJS server that serves static HTML and CSS files to the user without using Express** |
| --- | --- | --- |
| **DATE** |  |

# Aim:

# To create a NodeJS server that serves static HTML and CSS files to the user without using Express.

**Algorithm:**

**Step 1:**Start the Visual studio code by creating a new folder in desktop or any other application.

**Step 2:** Create a new file for Javascript(sample.js). Create a new file for HTML(sample.html).

**Step 3:** Using the terminal menu, comment node filename.js Server will be listening after running the Javascript. Open the web browser, Type the command localhost:3000.

**THEORY:**

* For creating server on Node JS we have used “http” module. For using any module in Node JS we have to use “require” module.   So fist we import the “http” module.

var http=require("http");

* Now we are creating a server. For creating the server we have to use ‘createServer ‘method of http module and this method takes 2 parametersing [request and response] as show below.

var server = http.createServer(function(request, response) {});

* After that we have to set the content type as plain text for sending response to the client. As shown in

var server = http.createServer(function(request, response) {

    response.writeHead(200, {

        'Content-Type': 'text/plain'

    });

});

Here you can set any response type like plain text or html etc.

* Now for sending response to client we have to use response.write() method. And finally you have to call response.end() method for ending the response.

var server = http.createServer(function(request, response) {

    response.writeHead(200, {

        'Content-Type': 'text/plain'

    });

    response.write("This is Test Message.");

    response.end();

});

* Now we have to start listening to this server; on any http post you can use any port which is available (not used by any other application in your computer.) Here I have taken 8082 port.  
    
  *server.listen(3000);*

**Step 4:** HTML page will be displayed.Stop the execution.

**Program:**

#Server.js

var http = require('http');

var fs = require('fs');

var path = require('path');

http.createServer(function (request, response) {

    console.log('request ', request.url);

    var filePath = '.' + request.url;

    if (filePath == './') {

        filePath = './index.html';

 }

    var extname = String(path.extname(filePath)).toLowerCase();

    var mimeTypes = {

        '.html': 'text/html',

        '.js': 'text/javascript',

        '.css': 'text/css',

        '.json': 'application/json',

        '.png': 'image/png',

        '.jpg': 'image/jpg',

        '.gif': 'image/gif',

        '.svg': 'image/svg+xml',

        '.wav': 'audio/wav',

        '.mp4': 'video/mp4',

        '.woff': 'application/font-woff',

        '.ttf': 'application/font-ttf',

        '.eot': 'application/vnd.ms-fontobject',

        '.otf': 'application/font-otf',

        '.wasm': 'application/wasm'

    };

    var contentType = mimeTypes[extname] || 'application/octet-stream';

    fs.readFile(filePath, function(error, content) {

        if (error) {

            if(error.code == 'ENOENT') {

                fs.readFile('./404.html', function(error, content) {

                    response.writeHead(404, { 'Content-Type': 'text/html' });

                    response.end(content, 'utf-8');

                });

            }

            else {

                response.writeHead(500);

                response.end('Sorry, check with the site admin for error: '+error.code+' ..\n');

            }  }

        else {

            response.writeHead(200, { 'Content-Type': contentType });

            response.end(content, 'utf-8');

        }

    });

}).listen(8125);

console.log('Server running at <http://127.0.0.1:8125/>');

#index.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></title>

</head>

<body>

    httpserver work without express

</body>

</html>

#404.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></title>

</head><body>

    httpserver work without express

</body></html>

**Output:**

    httpserver work without express

# Result

Thus, to create a NodeJS server that serves static HTML and CSS files to the user without using Express was done successfully.

| **ExNo.** | **4** | **Create NodeJS server using express that stores data from a form as a JSON file and display it in another page. The redirection page should be prepared using Handlebars** |
| --- | --- | --- |
| **DATE** |  |

# Aim:

# To Create NodeJS server using express that stores data from a form as a JSON file and display it in another page. The redirection page should be prepared using Handlebars.

**Algorithm:**

**Installing Express**

$ npm install express

**Handlebars:**

* Handlebars is a simple templating language.
* It uses a template and an input object to generate HTML or other text formats.
* Handlebars templates look like regular text with embedded Handlebars expressions.
* A handlebars expression is a {{, some contents, followed by a }}.
* When the template is executed, these expressions are replaced with values from an input object.

**JSON:**

* JSON stands for **J**ava**S**cript **O**bject **N**otation
* JSON is a lightweight format for storing and transporting data
* JSON is often used when data is sent from a server to a web page
* JSON is "self-desribing" and easy to understand

**Step 1:** start the visual studio code.

**Step 2**: As this is a full stack development we are going to front end and the back end.

**Step 3:**Read JSON data from disk

**Step 4:**Learn to use fs module to interact with the filesystem

**Step 5:**Persist data to a JSON file

**Step 6:**Use JSON.parse and JSON.stringify to convert data to and from JSON format

By the end of this tutorial you should be able to work with JSON files using Node’s built-in fs module

**Step 7:**To use handlebars in express, we need to store HTML code into a .hbs extension in the ‘views’ folder in the source directory as hbs looks for the pages in the views folder.

**Step 8:** The first thing we need to do in index.js file is to require the hbs module

| var express = require('express')  var hbs = require('hbs')  var app = express() |
| --- |

Now, we need to change the default view engine.

| app.set('view engine', 'hbs') |
| --- |

In case the views directory is undesirable, you can change the viewpath by the following command:

| app.set('views', <pathname>) |
| --- |

Now let us create a demo.hbs file in our views directory with the following content:

| <!DOCTYPE html>  <html>      <body>          <p>This is a Demo Page on localhost!</p>      </body>  </html> |
| --- |

Now, we render our webpage through express to the local server.

| app.get('/', (req, res)=>{      res.render('demo')  })    app.listen(3000) |
| --- |

Now, open your browser and type localhost:3000 on web address and verify the webpage at your server.

Now we will see how we can dynamically link the pages to server-side data.  
In the index.js, we declare a demo object, in practice, the object can be a result of the request body and/or database query.

| var demo = {      name : 'Rohan',      age : 26  }    app.get('/', (req, res)=>{       res.render('dynamic', {demo : demo})  }) |
| --- |

Here we send the demo object as a demo to our hbs page

**Program:**

**#app.js**

const express = require("express")

const path = require("path")

const app = express()

app.use(express.urlencoded())

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

    console.log("working")

    res.sendFile(path.join(\_\_dirname, '/index.html'))

})

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

    if(req.body.username=="root" && req.body.password =="root"){

        res.send("login success")

    }

    else{

    res.send("login failed")

}

})

app.listen(4000);

console.log("Server started at <http://localhost:4000>")

**#index.html**

<html>

  <head>

    <meta name="viewport" content="width=device-width, initial-scale=1">

    <link rel="stylesheet" href="/css/styles.css">

    <title>LOGIN PAGE</title>

<body>

    <h1>Login</h1>

    <form name="login" action="/" method="post">

      <div class="row">

        <label for="username">Username</label>

        <input type="text" name="username" />

      </div>

      <div class="row">

        <label for="password">Password</label>

        <input type="password" name="password" />

      </div>

      <input type="submit" class="button" value="Login" />

    </form>

    <style>

@import url('https://fonts.googleapis.com/css?family=Open+Sans&display=swap');

  body {

    font-family: 'Open Sans', sans-serif;

    background: #f9faff;

    color: #3a3c47;

    line-height: 1.6;

    display: flex;

    flex-direction: column;

    align-items: center;

    margin: 0;

    padding: 0;

  }

  h1 {

    margin-top: 48px;

  }

  form {

    background: #fff;

    max-width: 360px;

    width: 100%;

    padding: 58px 44px;

    border: 1px solid #e1e2f0;

    border-radius: 4px;

    box-shadow: 0 0 5px 0 rgba(42, 45, 48, 0.12);

    transition: all 0.3s ease;

  }

  .row {

    display: flex;

    flex-direction: column;

    margin-bottom: 20px;

  }

  .row label {

    font-size: 13px;

    color: #8086a9;

  }

  .row input {

    flex: 1;

    padding: 13px;

    border: 1px solid #d6d8e6;

    border-radius: 4px;

    font-size: 16px;

    transition: all 0.2s ease-out;

  }

  .row input:focus {

    outline: none;

    box-shadow: inset 2px 2px 5px 0 rgba(42, 45, 48, 0.12);

  }

  .row input::placeholder {

    color: #C8CDDF;

  }

  .button {

    width: 100%;

    padding: 12px;

    font-size: 18px;

    background: #15C39A;

    color: #fff;

    border: none;

    border-radius: 100px;

    cursor: pointer;

    font-family: 'Open Sans', sans-serif;

    margin-top: 15px;

    transition: background 0.2s ease-out; }

  .button:hover {

    background: #55D3AC;

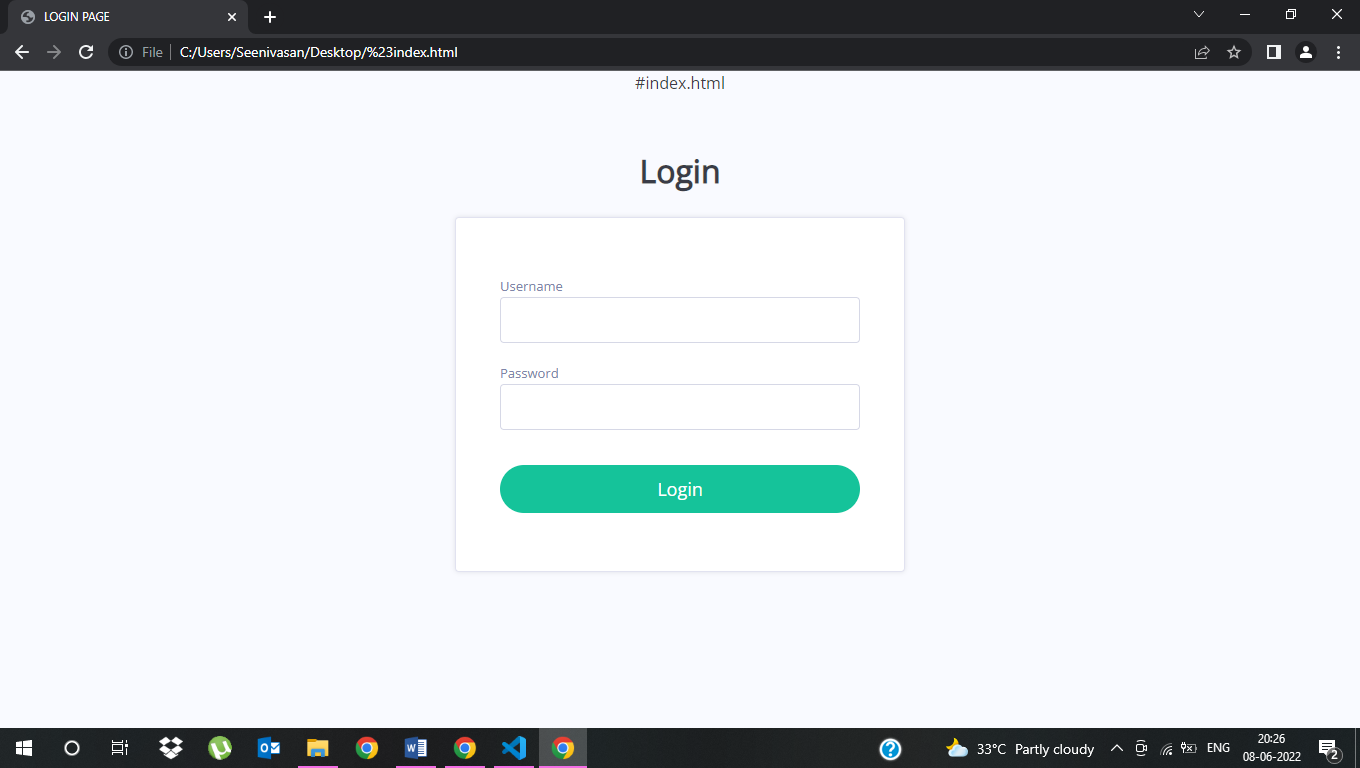
  }

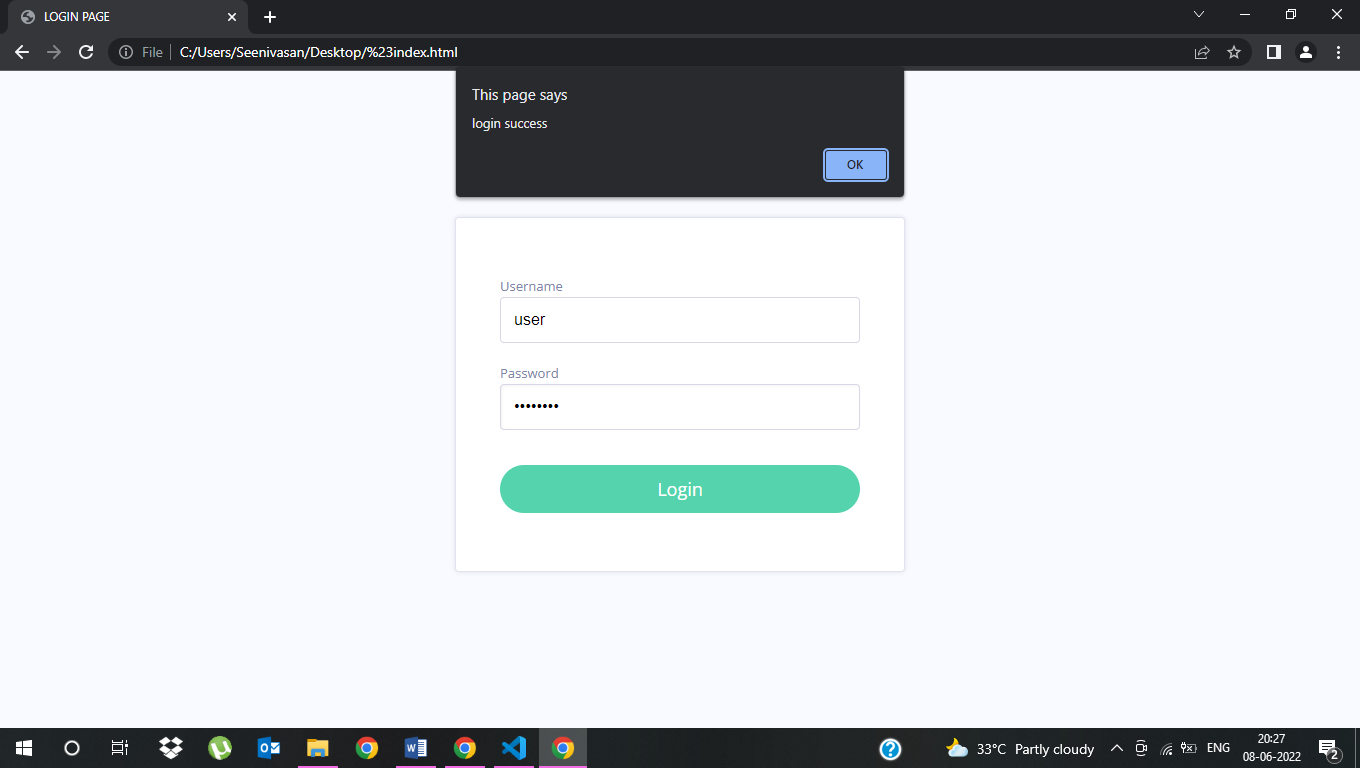
  </style>

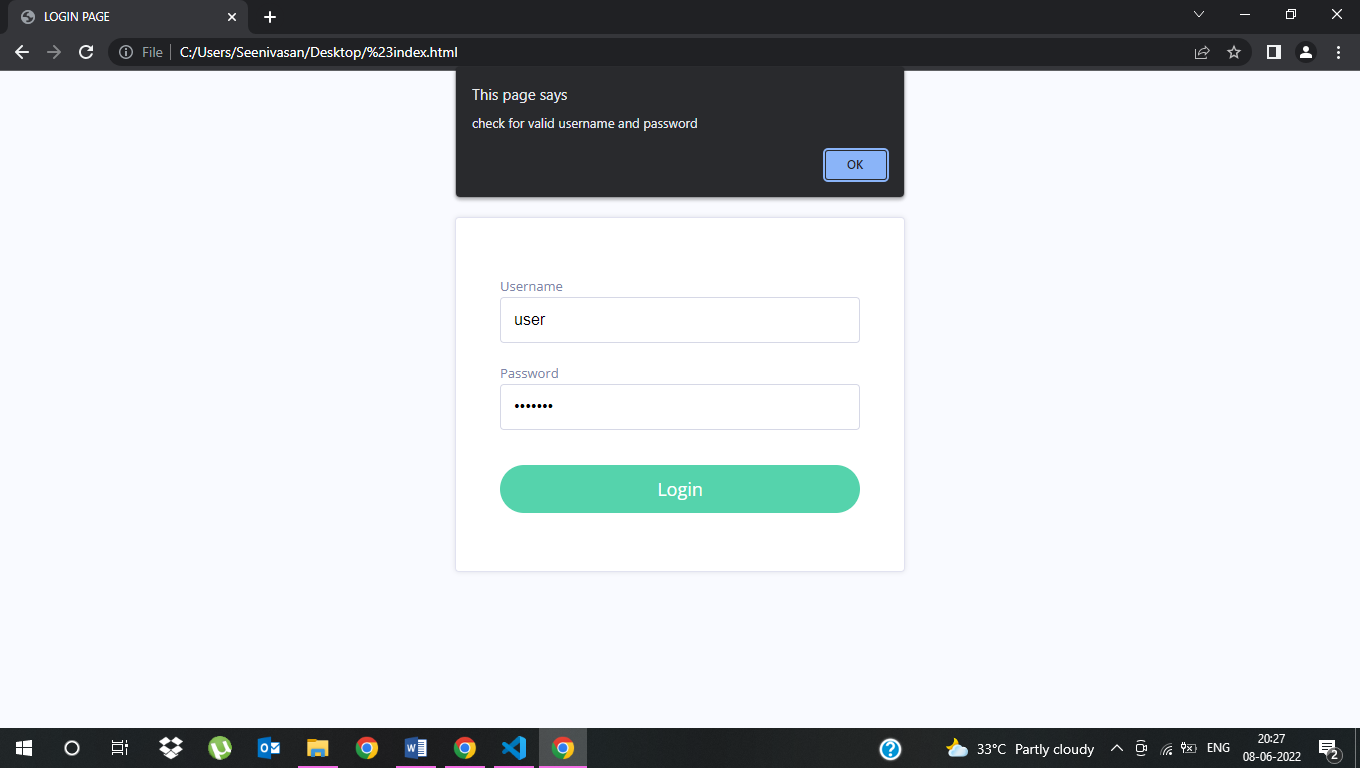
  </body>

</html>

**Output:**







# Result

# Thus,To Create NodeJS server using express that stores data from a form as a JSON file and display it in another page. The redirection page should be prepared using Handlebars was executed successfully.

.

| **ExNo.** | **5** | **Create NodeJS server using express that creates, reads, updates and deletes students details and store them in MongoDB database. The information about the user should be obtained from a HTML form.** |
| --- | --- | --- |
| **DATE** |  |

# Aim:

To Create a NodeJS server using Express that creates, reads, updates and deletes students' details and stores them in MongoDB database. The information about the user should be obtained from a HTML form.

# Algorithm:

[**MongoDB**](https://www.mongodb.com/)**is a database**. This is the place where you store information for your websites (or applications).

[**CRUD**](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete)**is an acronym for Create, Read, Update and Delete**. It is a set of operations we get servers to execute (POST, GET, PUT and DELETE requests respectively). This is what each operation does:

* **Create (POST)** - Make something
* **Read (GET)**- Get something
* **Update (PUT)** - Change something
* **Delete (DELETE)**- Remove something

**mongodb.connect()** method is the method of the MongoDB module of the Node.js which is used to connect the database with our Node.js Application.

**Syntax:**

mongodb.connect(path,callbackfunction)

**Installing module:**

npm install mongodb

Start the Visual studio code by creating a new folder in desktop or any other application.

**Step 1:**Create a new file for Javascript(index.js). Create a new file for HTML(index.html).

**Step 2:**Using the terminal menu, comment npm init(just provide enter and type yes)

**Step 3:**Using the terminal menu, npm install express for installing the express framework.

**Step 4:**Using the terminal menu, npm install mongodb for installing the hbs template engine.For running the code specify node index.js. Server will be listening after running the Javascript.

**Step 5:**Open the web browser, Type the command localhost:3000.

**Step 6:**HTML page will be displayed.

**Step 7:**Stop the execution.

**PROGRAM:**

**index.js**

const express = require("express");

const bodyParser = require("body-parser");

var MongoClient = require('mongodb').MongoClient;

const app = express();

app.use(bodyParser.urlencoded({ 'extended': true}));

var url = "mongodb://localhost:27017/mydb";

function createDB() {

let conn;

MongoClient.connect(url, function(err, db)

{

if (err)

throw err;

console.log("Database created!");

});

}

function createCollection(collectionName) {

MongoClient.connect(url, function(err, db) {

if (err) throw err;

var dbo = db.db("mydb");

dbo.createCollection(collectionName, function(err, res) {

if (err)

throw err;

console.log("Collection created!");

db.close();

});

});

}

function insertData(data) {

MongoClient.connect(url, function(err, db) {

if (err) throw err;

var dbo = db.db("mydb");

dbo.collection("person").insertOne(data, function(err, res) {

if (err) throw err;

console.log("1 document inserted");

db.close();

});

});

}

function readData(data, callback) {

let result;

MongoClient.connect(url, function(err, db) {

if (err) throw err;

var dbo = db.db("mydb");

dbo.collection("person").findOne(data, function(err, res) {

if (err) {

throw err;

} else {

callback(res);

return res;

}

});

});

}

function updateData(id, newvalues) {

MongoClient.connect(url, function(err, db) {

if (err) throw err;

var dbo = db.db("mydb");

dbo.collection("person").updateOne(id, newvalues, function(err, res) {

if (err) throw err;

console.log("1 document updated");

db.close();

});

});

}

// insertData({name:"ram",age:22});

// updateData({name:"ram"},  {$set:{age:21}});

function deleteData(myquery) {

MongoClient.connect(url, function(err, db) {

if (err) throw err;

var dbo = db.db("mydb");

dbo.collection("person").deleteOne(myquery, function(err, obj) {

if (err) throw err;

console.log("1 document deleted");

db.close();

});

});

}

// deleteData({name:"kumar"});

// function waitforData(data){

//

//   ret

// }

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

res.sendFile(\_\_dirname + "/index.html");

});

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

const data = req.body;

// {

//   name: "ram",

//   age:"22"

// }

insertData(data);

res.redirect("/");

});

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

const data = req.body;

console.log(data);

readData(data, (result) => {

res.send(` <h2> Name : ${result.name}  <br/> Age: ${result.age} </h2>`);

});

})

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

const data = req.body;

updateData({name: data.name}, { $set: {name: data.upname,age: data.upage}});

res.redirect("/");

})

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

const data = req.body;

deleteData(data);

res.redirect("/");

});

app.listen(3000, () => {

console.log("Server Listening in port 3000");

})

**index.html**

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title>CRUD Operations </title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

<style media="screen">

h1{

text-align: center;

}

.container{

padding: 5% 10%;

margin: auto;

background-color: #B7CADB;

}

</style>

</head>

<body>

<h1>CRUD Operations mongodb</h1>

<div class="container">

<div class="container">

<h3>Insert Data</h3>

<form action="/insert" method="post">

<div class="form-group">

<label for="search">Enter Name</label>

<input type="text" name="name" id="word" class="form-control" value="">

</div>

<div class="form-group">

<label for="search">Enter Age </label>

<input type="text" name="age" id="word" class="form-control" value="">

</div>

<br>

<button  type="submit"  class="btn btn-lg btn-primary"onclick="createReq()">Submit</button>

</form>

</div>

<div class="container">

<h3>Read Data</h3>

<form action="/read" method="post">

<div class="form-group">

<label for="search">Enter Name</label>

<input type="text" name="name" id="word" class="form-control" value="">

</div>

<br>

<button  type="submit"  class="btn btn-lg btn-primary"onclick="createReq()">Submit</button>

</form>

</div>

<div class="container">

<h3>Update Data</h3>

<form action="/update" method="post">

<div class="form-group">

<label for="search">Enter Name</label>

<input type="text" name="name" id="word" class="form-control" value="">

</div>

<div class="form-group">

<label for="search">Enter Updated Name</label>

<input type="text" name="upname" id="word" class="form-control" value="">

</div>

<div class="form-group">

<label for="search">Enter Updated Age</label>

<input type="text" name="upage" id="word" class="form-control" value="">

</div>

<br>

<button  type="submit"  class="btn btn-lg btn-primary"onclick="createReq()">Submit</button>

</form>

</div>

<div class="container">

<h3>Delete Data</h3>

<form action="/delete" method="post">

<div class="form-group">

<label for="search">Enter Name</label>

<input type="text" name="name" id="word" class="form-control" value="">

</div>

<br>

<button  type="submit"  class="btn btn-lg btn-primary"onclick="createReq()">Submit</button>

</form>

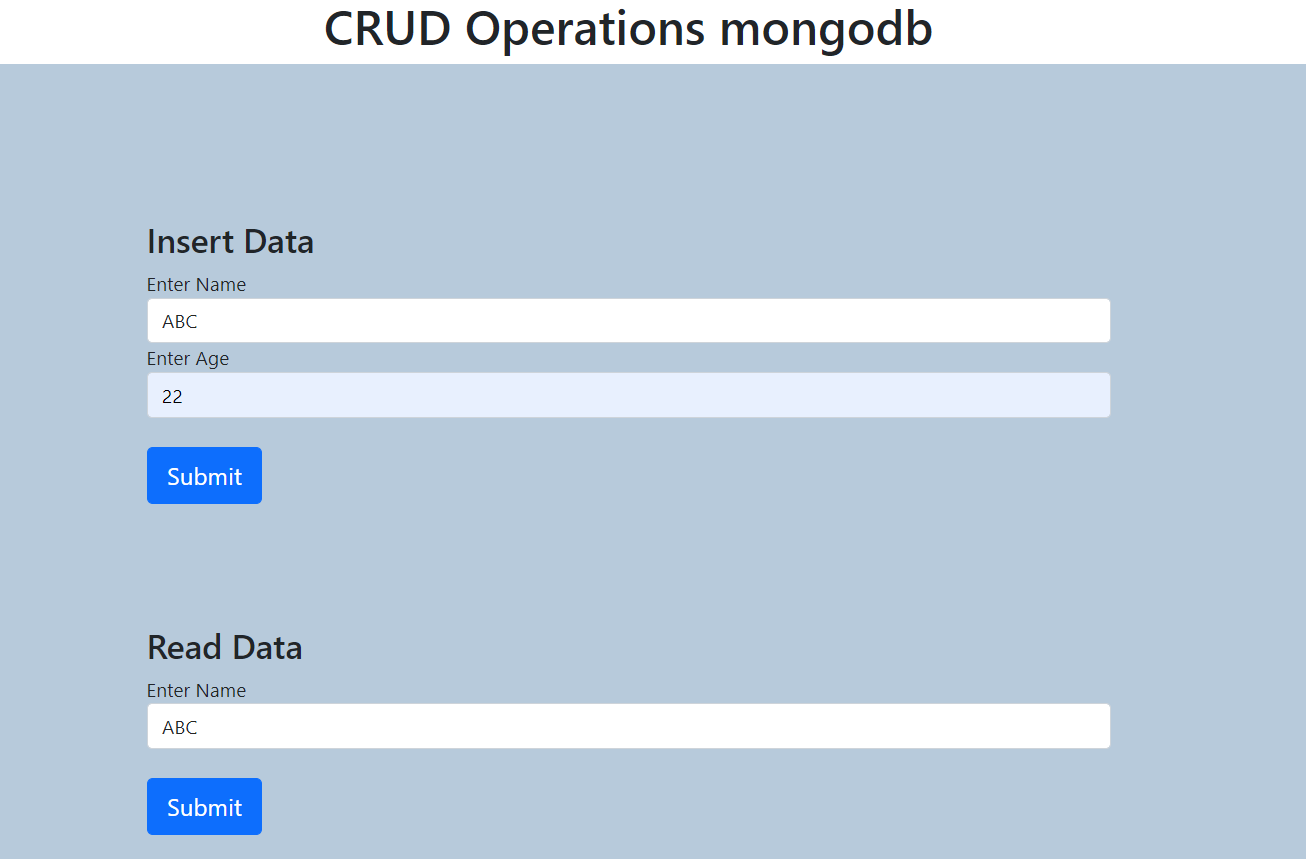
</div>

</div>

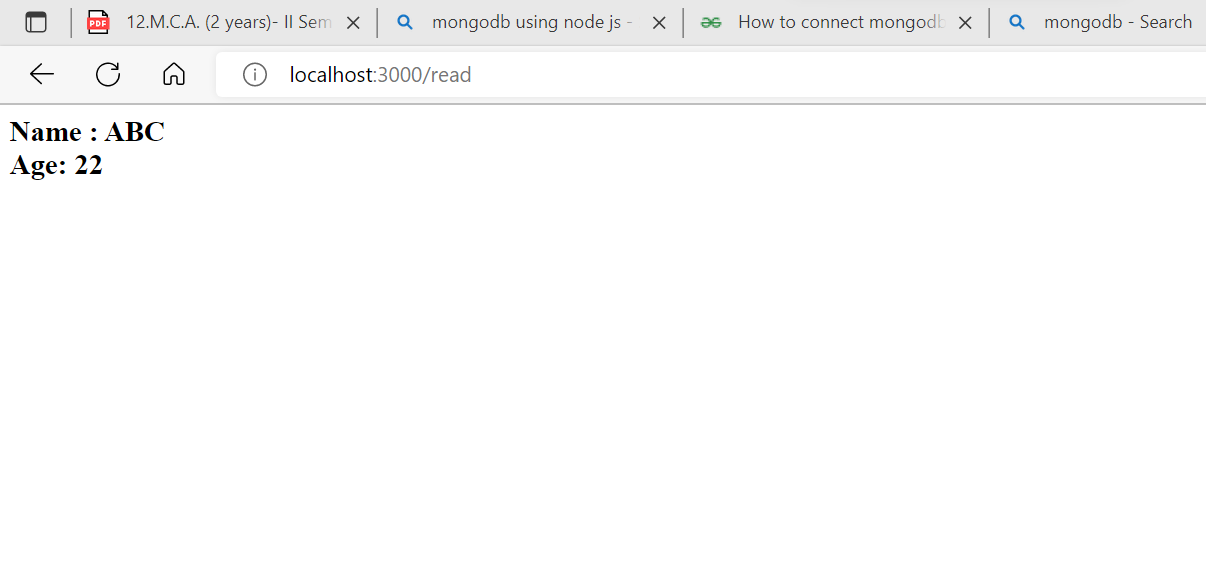
</body>

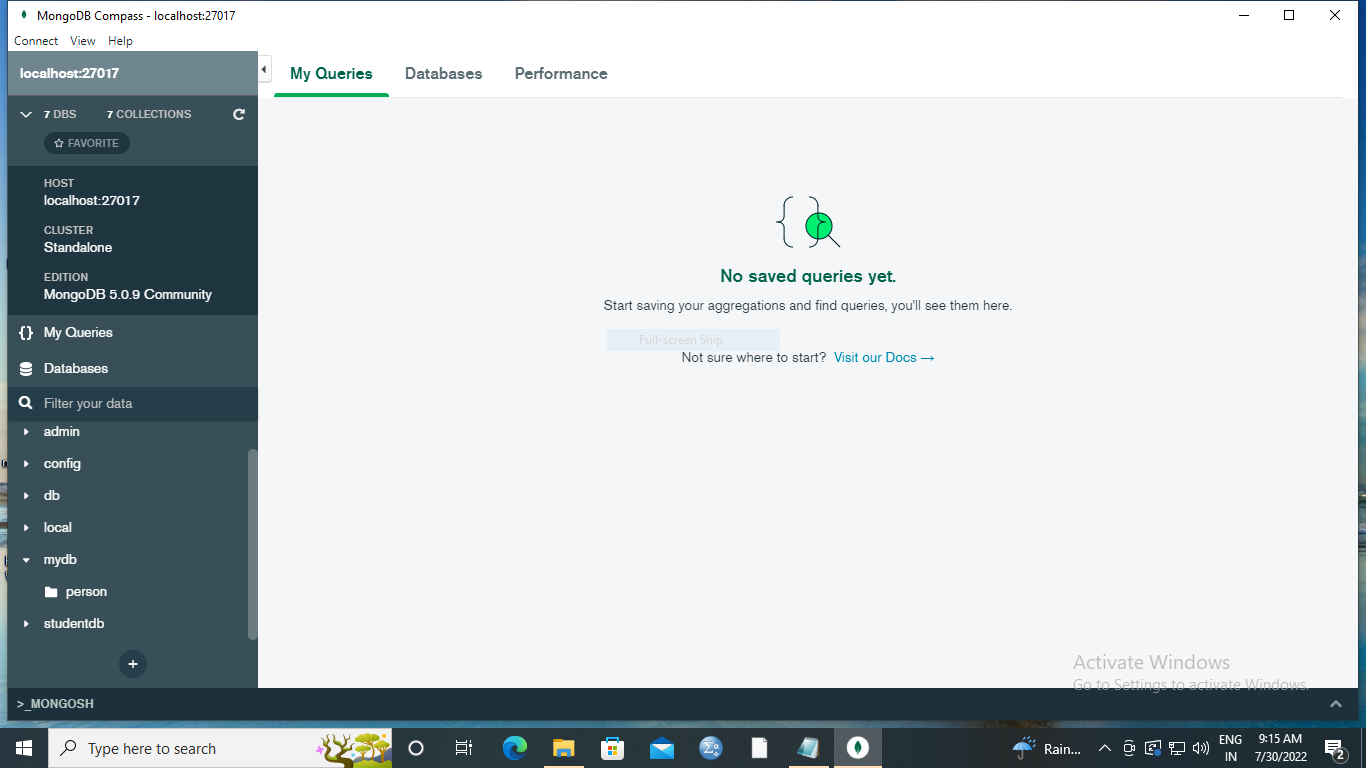
</html>

**OUTPUT**



**Read data:**

****

****

**RESULT:**

Thus, creating a NodeJS server using Express that creates, reads, updates and deletes students' details and stores them in MongoDB database obtained using HTML form.

| **ExNo.** | **6** | **Create NodeJS server that creates, reads, updates and deletes event details and stores them in MySQL database. The information about the user should be obtained from a HTML form.** |
| --- | --- | --- |
| **DATE** |  |

# Aim:

To Create a NodeJS server that creates, reads, updates and deletes event details and stores them in Mysql database. The information about the user should be obtained from a HTML form.

# Algorithm:

**MySQL is a database**. **MySQL is the most widely adopted open source relational database** and serves as the primary relational data store for many popular websites, applications, and commercial products.

[**CRUD**](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete)**is an acronym for Create, Read, Update and Delete**. It is a set of operations we get servers to execute (POST, GET, PUT and DELETE requests respectively). This is what each operation does:

* **Create (POST)** - Make something
* **Read (GET)**- Get something
* **Update (PUT)** - Change something
* **Delete (DELETE)**- Remove something

Start the Visual studio code by creating a new folder in desktop or any other application.

**Step 1:** Create a new file for Javascript(index.js). Create a new file for HTML(index.html).

**Step 2:** Using the terminal menu, comment npm init(just provide enter and type yes)

**Step 3:** Install mysql compass for running the code specify node index.js. Server will be listening after running the Javascript.

**Step 4:** Open the web browser, Type the command localhost:3000.

**Step 5:** HTML page will be displayed.

**Step 6:** Stop the execution.

**PROGRAM:**

**app.js**

const express = require("express");

const bodyParser = require("body-parser");

const mysql = require("mysql");

const app = express();

app.use(bodyParser.urlencoded({

  'extended': true

}));

var connection = mysql.createConnection({

  host: '127.0.0.1',

  port: 3306,

  user: 'root',

  password: 'root',

  database: 'person' // after creation of DB we can use the name here

});

function createDB() {

  connection.query("CREATE DATABASE Person", function (err, result) {

    if (err) throw err;

    console.log("Database created");

  });

}

function createTable() {

  connection.query("CREATE TABLE users(NAME VARCHAR(20), AGE INT )", function (err, result) {

    if (err) throw err;

    console.log("Table created");

  });

}

connection.connect((err) => {

  if (err) {

    console.log(err);

  }

  else {

    console.log("connected");

  }

});

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

  res.sendFile(\_\_dirname + "/index.html");

});

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

  const data = req.body;

  let sql = `INSERT INTO USERS VALUES( '${data.name}',${data.age})`;

  connection.query(sql, function (err, result) {

    if (err) throw err;

    console.log("1 record inserted");

  });

  res.redirect("/");

});

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

  const data = req.body;

  let sql = `SELECT \* FROM USERS WHERE NAME='${data.name}'`;

  connection.query(sql, (err, rows, col) => {

    if (err) {

      res.write(err);

    }

    else {

      let r = JSON.stringify(rows);

      res.write(r);

    }

    res.send();

  })

})

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

  const data = req.body;

  let sql = `UPDATE USERS SET NAME="${data.upname}", age= '${data.upage}' WHERE NAME = '${data.name}'`;

  connection.query(sql, function (err, result) {

    if (err) throw err;

    console.log("1 record modified");

  });

  res.redirect("/");

})

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

  const data = req.body;

  let sql = `DELETE FROM USERS WHERE NAME='${data.name}'`;

  connection.query(sql, function (err, result) {

    if (err) throw err;

    console.log("1 record deleted");

  });

  res.redirect("/");

});

app.listen(5000, () => {

  console.log("Server Listening in port 3000");

})

**index.html**

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title>CRUD Operations </title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

<style media="screen">

h1{

text-align: center;

}

.container{

padding: 5% 10%;

margin: auto;

background-color: #B7CADB;

}

</style>

</head>

<body>

<h1>CRUD Operations mongodb</h1>

<div class="container">

<div class="container">

<h3>Insert Data</h3>

<form action="/insert" method="post">

<div class="form-group">

<label for="search">Enter Name</label>

<input type="text" name="name" id="word" class="form-control" value="">

</div>

<div class="form-group">

<label for="search">Enter Age </label>

<input type="text" name="age" id="word" class="form-control" value="">

</div>

<br>

<button  type="submit"  class="btn btn-lg btn-primary"onclick="createReq()">Submit</button>

</form>

</div>

<div class="container">

<h3>Read Data</h3>

<form action="/read" method="post">

<div class="form-group">

<label for="search">Enter Name</label>

<input type="text" name="name" id="word" class="form-control" value="">

</div>

<br>

<button  type="submit"  class="btn btn-lg btn-primary"onclick="createReq()">Submit</button>

</form>

</div>

<div class="container">

<h3>Update Data</h3>

<form action="/update" method="post">

<div class="form-group">

<label for="search">Enter Name</label>

<input type="text" name="name" id="word" class="form-control" value="">

</div>

<div class="form-group">

<label for="search">Enter Updated Name</label>

<input type="text" name="upname" id="word" class="form-control" value="">

</div>

<div class="form-group">

<label for="search">Enter Updated Age</label>

<input type="text" name="upage" id="word" class="form-control" value="">

</div>

<br>

<button  type="submit"  class="btn btn-lg btn-primary"onclick="createReq()">Submit</button>

</form>

</div>

<div class="container">

<h3>Delete Data</h3>

<form action="/delete" method="post">

<div class="form-group">

<label for="search">Enter Name</label>

<input type="text" name="name" id="word" class="form-control" value="">

</div>

<br>

<button  type="submit"  class="btn btn-lg btn-primary"onclick="createReq()">Submit</button>

</form>

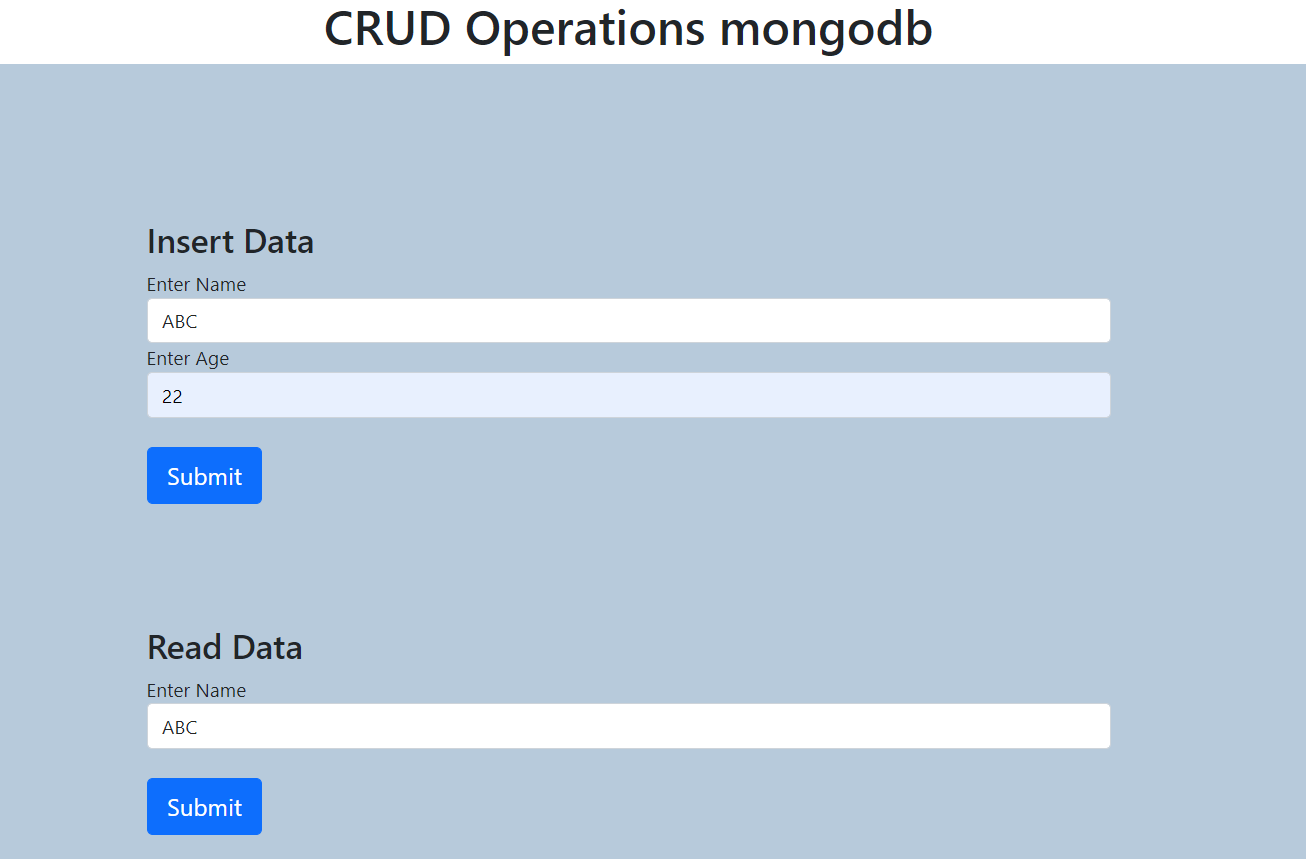
</div>

</div>

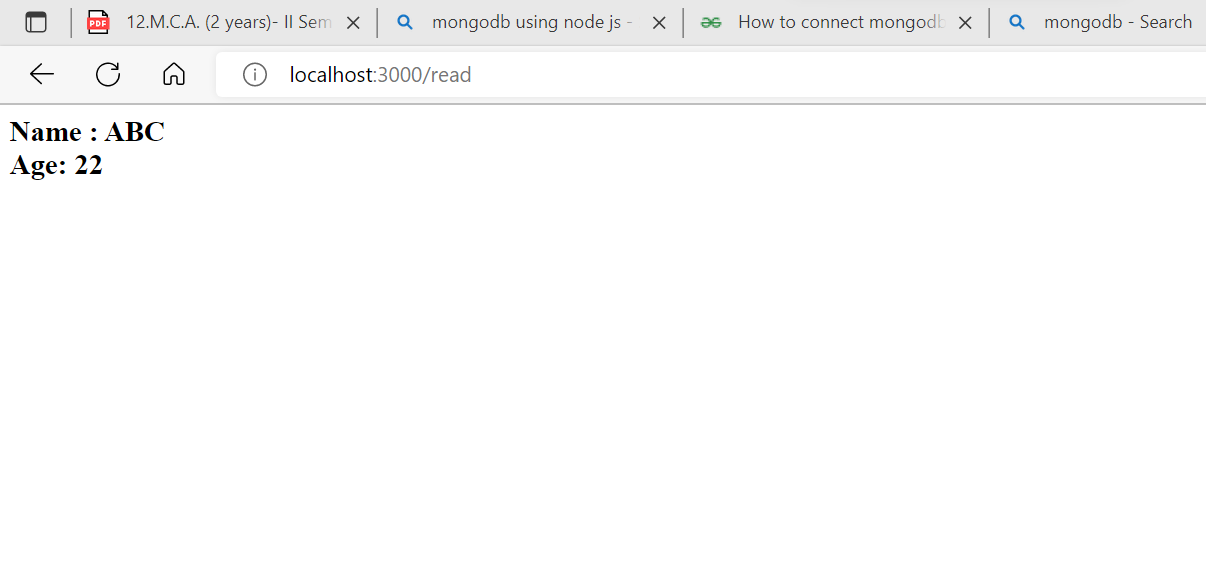
</body>

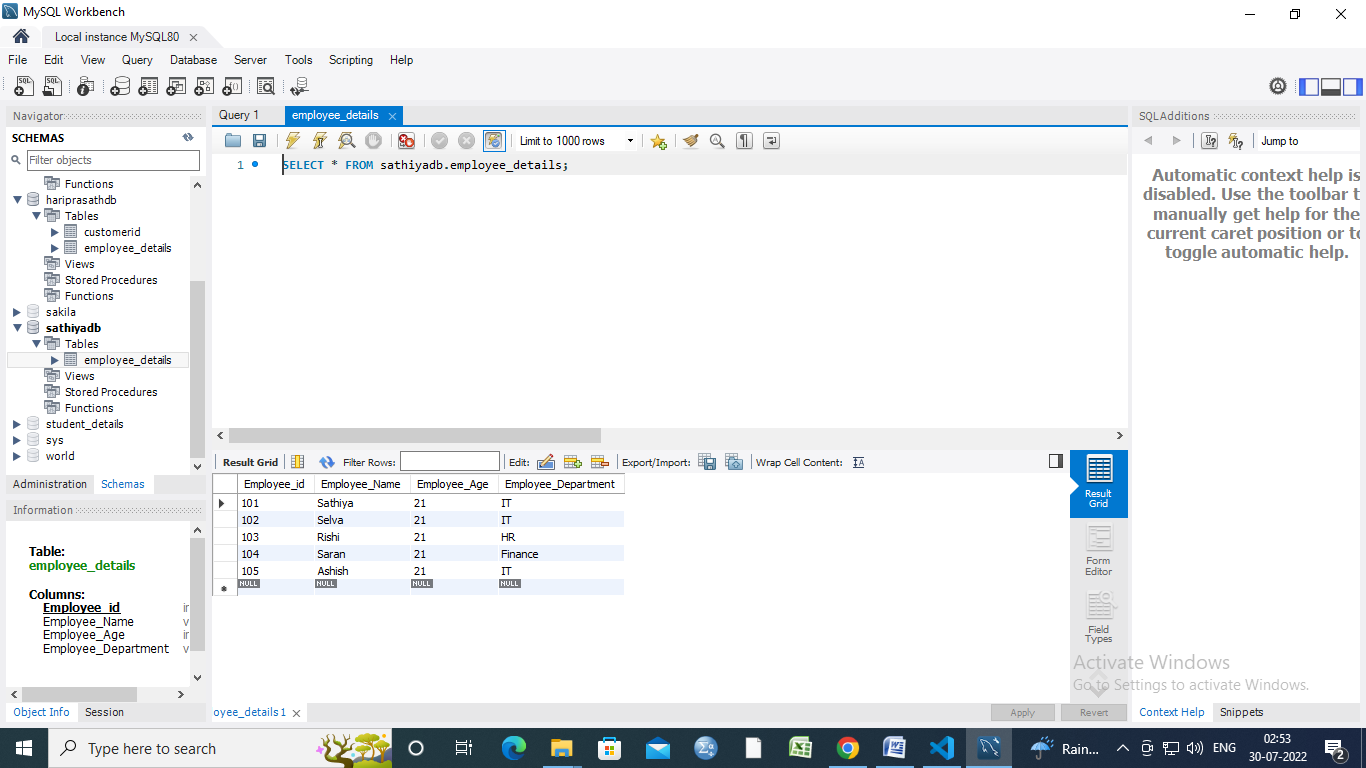
</html>

**OUTPUT**



**Read data:**

****

****

**RESULT:**

Thus, creating a NodeJS server that creates, reads, updates and deletes event details and stores them in Mysql database obtained using HTML form.

| **ExNo.** | **7** | **Create a counter using ReactJS** |
| --- | --- | --- |
| **DATE** |  |

# Aim:

# To create a counter using ReactJS.

**React** is a front-end, open-source JavaScript library that is used to create interactive UI. It is developed and maintained by Facebook. It can be used for the development of single-page and mobile applications.

**Initial Setup:**The npx is a CLI tool used to install and manage dependencies in the npm registry. NPX comes pre-bundled with npm 5.2+, else we can install it using the following command:

**npm i -g npx // -g flag indicates global installation**

**Algorithm:**

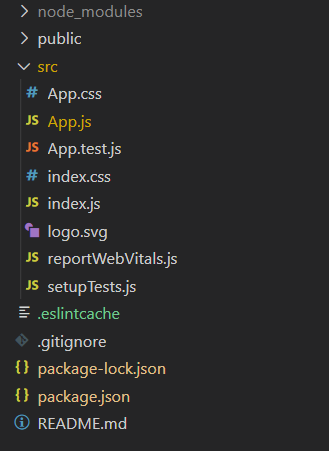
**Step 1:** Create a React application using the following command:

npx create-react-app counter

**Step 2:** After creating your project folder i.e. counter, move to it using the following command:

cd counter

**Project Structure:** It will look like the following.



*Project Structure*

**Step 3**: Run the application using the following command from the root directory of the project:

npm start

**Program:**

import React, { useState } from "react";

// Importing app.css is css file to add styling

import "./App.css";

const App = () => {

// Counter is a state initialized to 0

const [counter, setCounter] = useState(0)

// Function is called everytime increment button is clicked

const handleClick1 = () => {

// Counter state is incremented

setCounter(counter + 1)

}

// Function is called everytime decrement button is clicked

const handleClick2 = () => {

// Counter state is decremented

setCounter(counter - 1)

}

return (

<div style={{

display: 'flex',

flexDirection: 'column',

alignItems: 'center',

justifyContent: 'center',

fontSize: '300%',

position: 'absolute',

width: '100%',

height: '100%',

top: '-15%',

}}>

Counter App

<div style={{

fontSize: '120%',

position: 'relative',

top: '10vh',

}}>

{counter}

</div>

<div className="buttons">

<button style={{

fontSize: '60%',

position: 'relative',

top: '20vh',

marginRight: '5px',

backgroundColor: 'green',

borderRadius: '8%',

color: 'white',

}}

onClick={handleClick1}>Increment</button>

<button style={{

fontSize: '60%',

position: 'relative',

top: '20vh',

marginLeft: '5px',

backgroundColor: 'red',

borderRadius: '8%',

color: 'white',

}}

onClick={handleClick2}>Decrement</button>

</div>

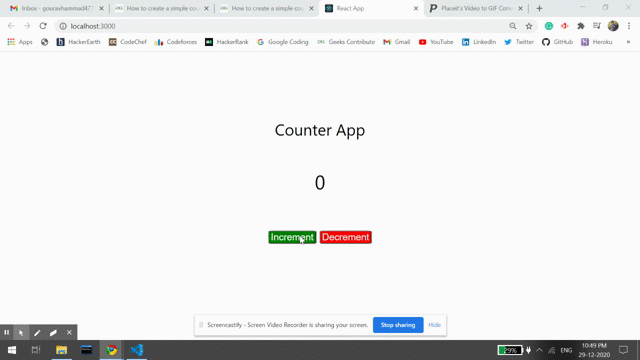
</div>

)

}

export default App

**Output:**



**Result**

Thus, to create a counter using ReactJS was successfully completed.

| **ExNo.** | **8** | **Create a Todo application using ReactJS. Store the data to a JSON file using a simple NodeJS server and retrieve the information from the same during page reloads** |
| --- | --- | --- |
| **DATE** |  |

# Aim:

# To Create a Todo application using ReactJS. Store the data to a JSON file using a simple NodeJS server and retrieve the information from the same during page reloads.

# React is a JavaScript library used to develop interactive user interfaces. It is managed by Facebook and a community of individual developers and companies. React mainly focuses on developing single-page web or mobile applications. here, we will create a todo app to understand the basics of **reactJS**.

**Modules required:**

* [npm](https://www.geeksforgeeks.org/node-js-npm-node-package-manager/)
* [React](https://www.geeksforgeeks.org/react-js-introduction-working/)
* React Bootstrap

npm install react-bootstrap bootstrap

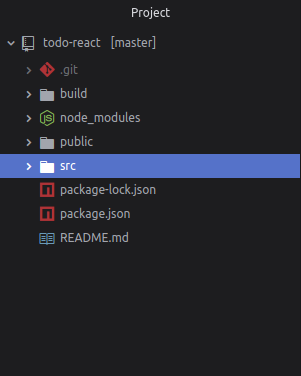
**Algorithm:**

**Step 1:** [NPX:](https://www.geeksforgeeks.org/what-are-the-differences-between-npm-and-npx/) It is a package runner tool that comes with npm 5.2+, npx is easy to use CLI tools. The npx is used for executing Node packages. It greatly simplifies a number of things one of which is checked/run a node package quickly without installing it locally or globally.

npx create-react-app todo-react

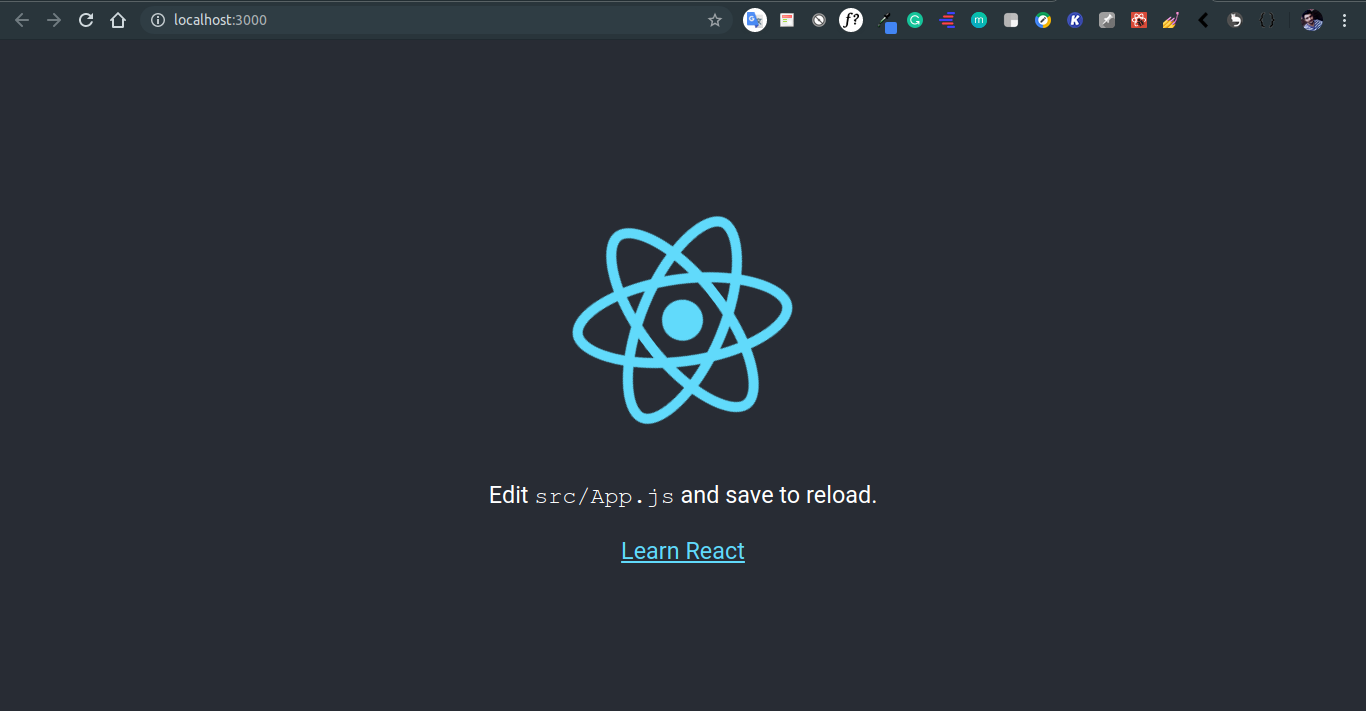
**Step 2:** Now, goto the folder

cd todo-react



**Step 3**: Start the server- Start the server by typing the following command in terminal:

npm start

open **http://localhost:3000/**  


* Change directory to src:

cd src

* Delete everything inside the directory

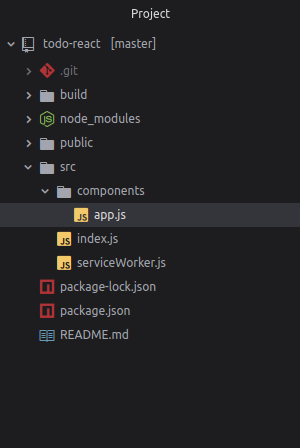
rm \*

* Now create **index.js** file

touch index.js

* This file will render our app to HTML file which is in **public folder**. Also create a folder name components with file name **app.js**

mkdir components && cd components && touch app.js

* **app.js** will contain our To-Do app:  
  
* **Save all files and start the server:**

npm start

* **Output:** Open **http://localhost:3000/** in browser

**Program:**

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import App from './components/app';

import 'bootstrap/dist/css/bootstrap.min.css';

ReactDOM.render(<App />, document.getElementById('root'));

**app.js**

import React, {Component} from 'react';

// Bootstrap for react

import Container from 'react-bootstrap/Container';

import Row from 'react-bootstrap/Row';

import Col from 'react-bootstrap/Col';

import Button from 'react-bootstrap/Button';

import InputGroup from 'react-bootstrap/InputGroup';

import FormControl from 'react-bootstrap/FormControl';

import ListGroup from 'react-bootstrap/ListGroup';

class App extends Component {

constructor(props) {

super(props);

// Setting up state

this.state = {

userInput : "",

list:[]

}

}

// Set a user input value

updateInput(value){

this.setState({

userInput: value,

});

}

// Add item if user input in not empty

addItem(){

if(this.state.userInput !== '' ){

const userInput = {

// Add a random id which is used to delete

id : Math.random(),

// Add a user value to list

value : this.state.userInput

};

// Update list

const list = [...this.state.list];

list.push(userInput);

// reset state

this.setState({

list,

userInput:""

});

}

}

// Function to delete item from list use id to delete

deleteItem(key){

const list = [...this.state.list];

// Filter values and leave value which we need to delete

const updateList = list.filter(item => item.id !== key);

// Update list in state

this.setState({

list:updateList,

});

}

render(){

return(<Container>

<Row style={{

display: "flex",

justifyContent: "center",

alignItems: "center",

fontSize: '3rem',

fontWeight: 'bolder',

}}

>TODO LIST

</Row>

<hr/>

<Row>

<Col md={{ span: 5, offset: 4 }}>

<InputGroup className="mb-3">

<FormControl

placeholder="add item . . . "

size="lg"

value = {this.state.userInput}

onChange = {item => this.updateInput(item.target.value)}

aria-label="add something"

aria-describedby="basic-addon2"

/>

<InputGroup.Append>

<Button

variant="dark"

size="lg"

onClick = {()=>this.addItem()}

>

ADD

</Button>

</InputGroup.Append>

</InputGroup>

</Col>

</Row>

<Row>

<Col md={{ span: 5, offset: 4 }}>

<ListGroup>

{/\* map over and print items \*/}

{this.state.list.map(item => {return(

<ListGroup.Item variant="dark" action

onClick = { () => this.deleteItem(item.id) }>

{item.value}

</ListGroup.Item>

)})}

</ListGroup>

</Col>

</Row>

</Container>

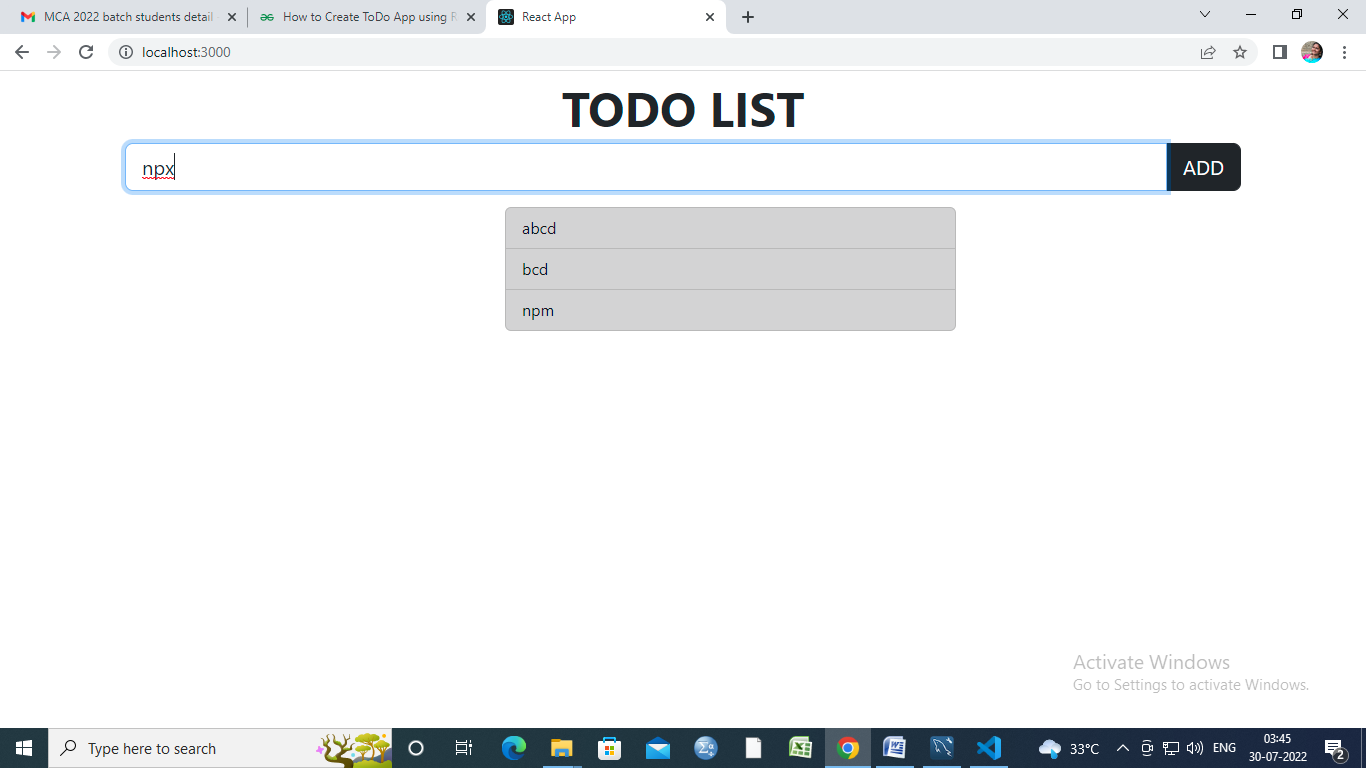
);

}

}

export default App;

**Output:**

****

**Result**

Thus, to To **Create a Todo application using ReactJS. Store the data to a JSON file using a simple NodeJS server and retrieve the information from the same during page reloads** completed successfully.