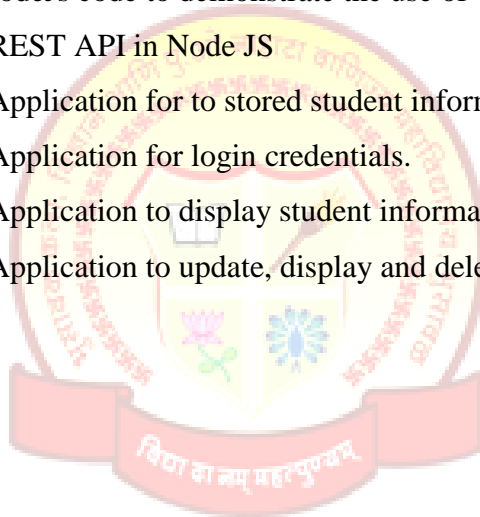


**BCA 607(A) Lab Web Development Technology - IV**  
**(React Js and Node JS)**  
**W.E.F. 2024-25**

**Assignments**

1. Write ReactJs code to use all the states in in the created Application.
2. Write ReactJs code for Client-side form validation.
3. Write ReactJs code for Applying form components.
4. Write ReactJs code to create student Registration Form.
5. Write ReactJs code to create Simple Login Form.
6. Write ReactJs Create a Single Page Application.
7. Write ReactJs / NodeJs code to Applying Routing.
8. Write ReactJs / NodeJs code to demonstrate the use of POST Method.
9. Write ReactJs/ NodeJs code to demonstrate the use of GET Method.
10. To demonstrate REST API in Node JS
11. Create Node JS Application for to stored student information in database.
12. Create Node JS Application for login credentials.
13. Create Node JS Application to display student information.
14. Create Node JS Application to update, display and delete student information.

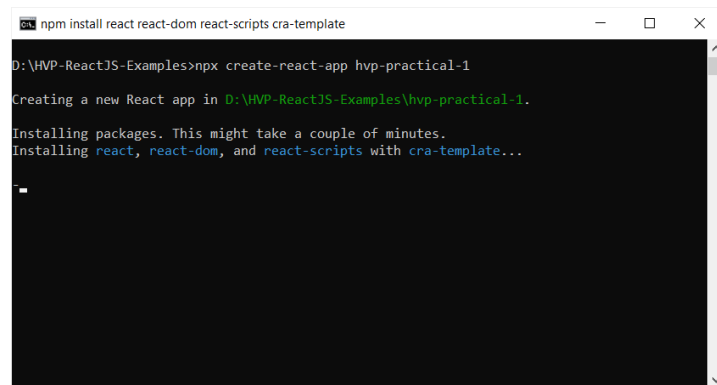


## Assignments – 1 Write ReactJs code to use all the states in in the created Application.

// class and functional components, and managing their states.

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

**`npx create-react-app <<hvp-practical-1>>`**



```
npm install react react-dom react-scripts cra-template
D:\HVP-ReactJS-Examples>npx create-react-app hvp-practical-1
Creating a new React app in D:\HVP-ReactJS-Examples\hvp-practical-1.
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
```

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

**`cd <<hvp-practical-1>>`**

### Step 3: Create the Main Application Component

Modify `App.js`: This will be the main component that combines class and functional components.

```
// src/App.js
import React from "react";
import ClassComponent from "./ClassComponent";
import FunctionalComponent from "./FunctionalComponent";
function App() {
  return (
    <div>
      <h1>Pract – I ReactJS State Management Examples</h1>
      <ClassComponent />
      <FunctionalComponent />
    </div>
  );
}
export default App;
```

### Step 4: Create the Class Component

Create `ClassComponent.js`: This component will manage its state using a class.

```
// src/ClassComponent.js
```

```

import React from "react";
class ClassComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      message: "Welcome to TYBCA Students",
      count: 0,
    };
  }
  toggleMessage = () => {
    this.setState({
      message:
        this.state.message === "Welcome to TYBCA Students"
          ? "Web Developments with ReactJS and NodeJS"
          : "Welcome to TYBCA Students",
    });
  };
  incrementCount = () => {
    this.setState((prevState) => ({
      count: prevState.count + 1,
    }));
  };
  render() {
    return (
      <div>
        <h2>Class Component</h2>
        <h3>{this.state.message}</h3>
        <button onClick={this.toggleMessage}>Toggle Message</button>
        <h3>Count: {this.state.count}</h3>
        <button onClick={this.incrementCount}>Increment Count</button>
      </div>
    );
  }
}
export default ClassComponent;

```

### Step 5: Create the Functional Component

Create `FunctionalComponent.js`: This component will manage its state using hooks.

```
// src/FunctionalComponent.js
import React, { useState, useEffect } from "react";
function FunctionalComponent() {
  const [message, setMessage] = useState("Welcome to TYBCA Students");
  const [count, setCount] = useState(0);
  const toggleMessage = () => {
    setMessage((prevMessage) =>
      prevMessage === "Welcome to TYBCA Students"
        ? "Web Developments with ReactJS and NodeJS"
        : "Welcome to TYBCA Students"
    );
  };
  const incrementCount = () => {
    setCount((prevCount) => prevCount + 1);
  };
  // Example of useEffect for side effects
  useEffect(() => {
    document.title = `Count: ${count}`;
  }, [count]);
  return (
    <div>
      <h2>Functional Component</h2>
      <h3>{ message}</h3>
      <button onClick={toggleMessage}>Toggle Message</button>
      <h3>Count: {count}</h3>
      <button onClick={incrementCount}>Increment Count</button>
    </div>
  );
}
export default FunctionalComponent;
```

### Step 6: Update the `index.js` File

Modify `index.js`: Ensure your main file imports and renders the `App` component.

```
// src/index.js
import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from "./App";
import reportWebVitals from "./reportWebVitals";

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById("root")
);
reportWebVitals();
```

### Step 7: Update the `index.html` File

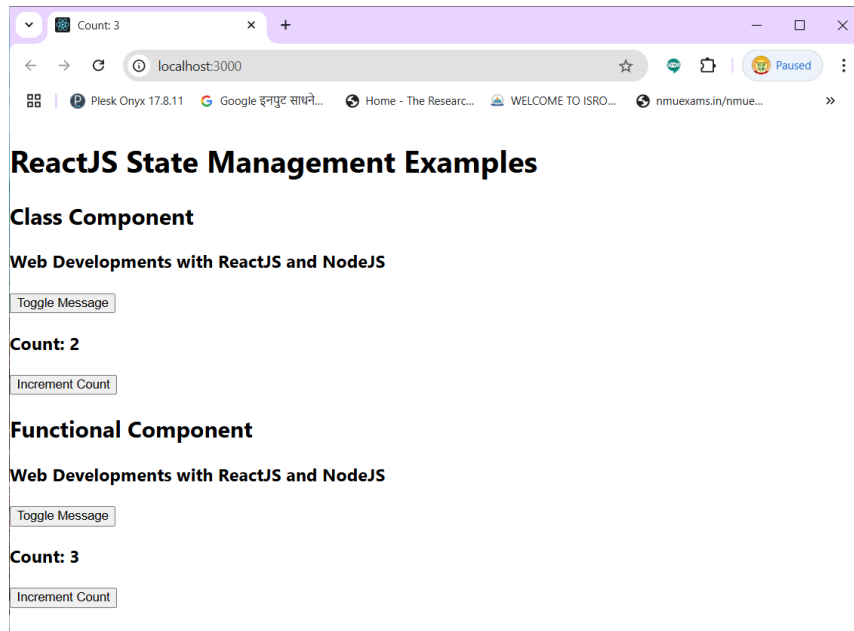
Ensure the `index.html` file in `public` folder contains the root element:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>ReactJS State Management Examples</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

**Step 8: Step to Run Application:** Run the application using the following command from the root directory of the project:

**npm start**

**Output:** Now open your browser and go to <http://localhost:3000/>, you will see the following output:



## Assignments – 2 Write ReactJs code for Client-side form validation.

// Client-side form validation using Custom Validation with State

```
import React, { useState } from 'react';

const FormWithValidation = () => {

  // Step 1: Set up state
  const [formData, setFormData] = useState({
    name: "",
    email: "",
  });

  const [errors, setErrors] = useState({});

  // Step 2: Handle input changes
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({
      ...formData,
      [name]: value,
    });
  };

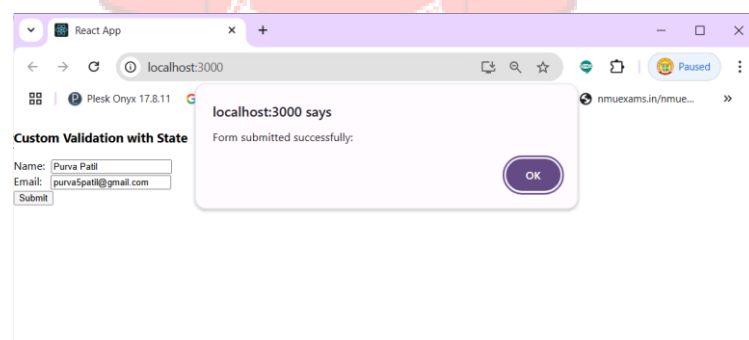
  // Step 3: Validate form data
  const validate = () => {
    const newErrors = {};
    if (!formData.name) {
      newErrors.name = 'Name is required';
    }
    if (!formData.email) {
      newErrors.email = 'Email is required';
    } else if (!/^\S+@\S+\.\S+$/.test(formData.email)) {
      newErrors.email = 'Email address is invalid';
    }
    return newErrors;
  };

  // Step 4: Handle form submission
  const handleSubmit = (e) => {
    e.preventDefault();
```

[illegible]



**Output:**

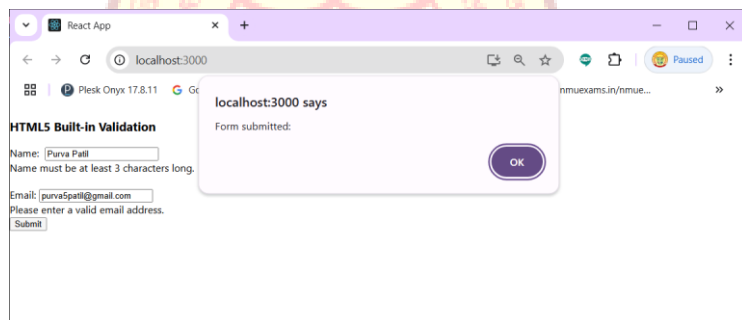
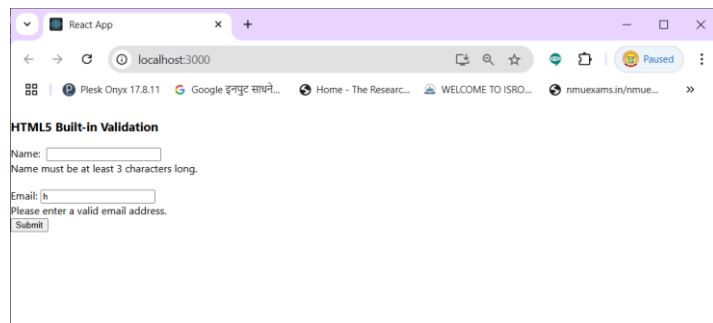


## // Client-side form validation using HTML5 Built-in Validation

```
import React from 'react';
const HTML5ValidationForm = () => {
  const handleSubmit = (e) => {
    e.preventDefault();
    const form = e.target;
    if (form.checkValidity() === false) {
      e.stopPropagation();
    } else {
      const formData = new FormData(form);
      const data = Object.fromEntries(formData.entries());
      alert('Form submitted:', data);
    }
    form.classList.add('was-validated');
  };
  return (
    <form onSubmit={handleSubmit} noValidate>
      <h3>HTML5 Built-in Validation</h3>
      <div>
        <label>
          Name: &nbsp;
          <input type="text" name="name" required minLength="3" />
          <div className="invalid-feedback">
            Name must be at least 3 characters long.
          </div>
        </label>
      </div>
      <br></br>
      <div>
        <label>
          Email:&nbsp;
          <input type="email" name="email" required />
          <div className="invalid-feedback">
            Please enter a valid email address.
          </div>
        </label>
      </div>
    </form>
  );
}
```

```
        </label>
      </div>
      <button type="submit">Submit</button>
    </form>
  );
};
export default HTML5ValidationForm;
```

## Output:



## // Client-side form validation using Third-Party Libraries

Before use this libraries first Ensure you have both formik and yup installed in your project using **npm install formik yup**

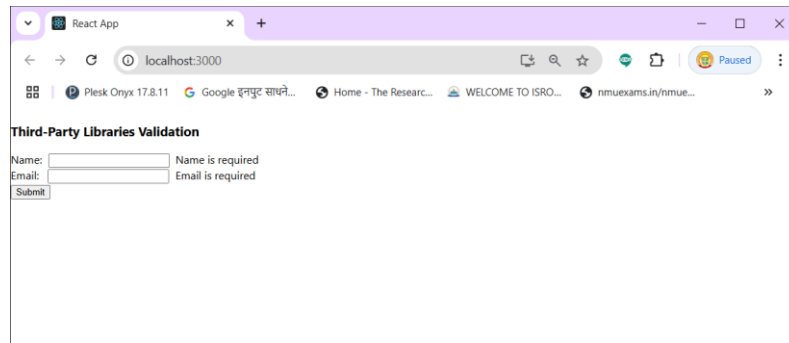
```
import React from 'react';
import { useFormik } from 'formik';
import * as Yup from 'yup';

const FormikYupValidationForm = () => {
  const formik = useFormik({
    initialValues: {
      name: "",
      email: "",
    },
    validationSchema: Yup.object({
      name: Yup.string()
        .min(3, 'Name must be at least 3 characters long')
        .required('Name is required'),
      email: Yup.string()
        .email('Invalid email address')
        .required('Email is required'),
    }),
    onSubmit: (values) => {
      alert('Form submitted:', values);
    },
  });

  return (
    <form onSubmit={formik.handleSubmit}>
      <h3>Third-Party Libraries Validation</h3>
      <div>
        <label>
          Name: &nbsp;
          <input
            type="text"
            name="name"
            value={formik.values.name}
            onChange={formik.handleChange}
          />
        </label>
      </div>
    </form>
  );
}
```

[illegible]

## Output:



React App

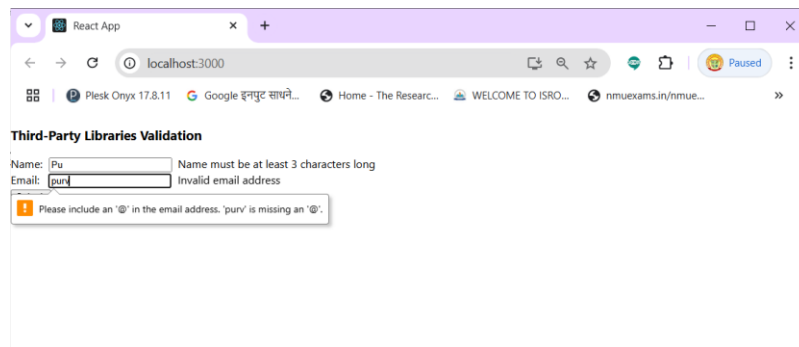
localhost:3000

Plesk Onyx 17.8.11 Google इन्फुट साधने... Home - The Researc... WELCOME TO ISRO... nmuexams.in/nmue...

### Third-Party Libraries Validation

Name:  Name is required

Email:  Email is required



React App

localhost:3000

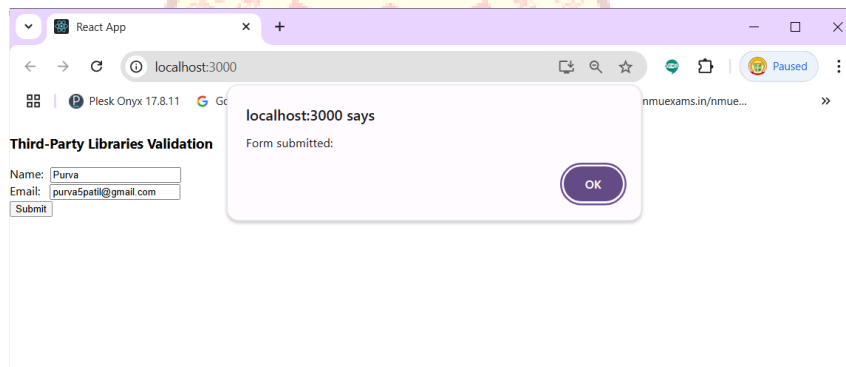
Plesk Onyx 17.8.11 Google इन्फुट साधने... Home - The Researc... WELCOME TO ISRO... nmuexams.in/nmue...

### Third-Party Libraries Validation

Name:  Name must be at least 3 characters long

Email:  Invalid email address

Please include an '@' in the email address. 'purn' is missing an '@'.



React App

localhost:3000

Plesk Onyx 17.8.11 Google इन्फुट साधने... Home - The Researc... WELCOME TO ISRO... nmuexams.in/nmue...

### Third-Party Libraries Validation

Name:

Email:

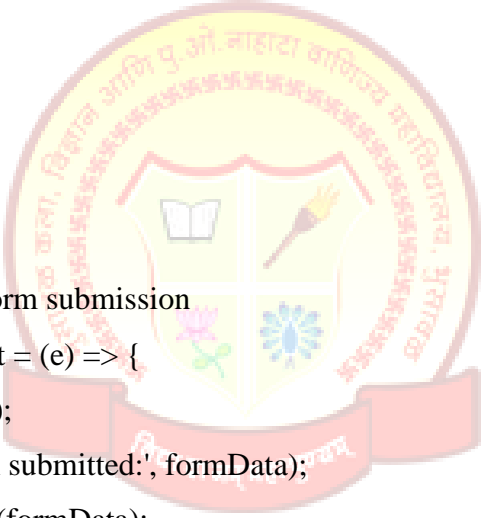
localhost:3000 says  
Form submitted:

### Assignments – 3 Write ReactJs code for Applying form components.

**// Write ReactJs code for Applying form components using Controlled Form**

```
import React, { useState } from 'react';

const MyForm = () => {
  // Step 1: Set up state
  const [formData, setFormData] = useState({
    name: "",
    email: "",
  });
  const [submittedData, setSubmittedData] = useState(null);
  // Step 2: Handle input changes
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({
      ...formData,
      [name]: value,
    });
  };
  // Step 3: Handle form submission
  const handleSubmit = (e) => {
    e.preventDefault();
    console.log('Form submitted:', formData);
    setSubmittedData(formData);
    // You can add further processing here (e.g., API calls)
  };
  return (
    <div>
      <form onSubmit={handleSubmit}>
        <div>
          <label>
            Name: &nbsp;
            <input
              type="text"
              name="name"
              value={formData.name}
            />
          </label>
        </div>
      </form>
    </div>
  );
};
```







## Output:

React App

localhost:3000

Plesk Onyx 17.8.11 Google इनपुट साधने...

Name:

Email:

**Submitted Data:**

Name: Harshal Patil

Email: harshal4patil@gmail.com



**// Write ReactJs code for Applying form components using Uncontrolled Form**

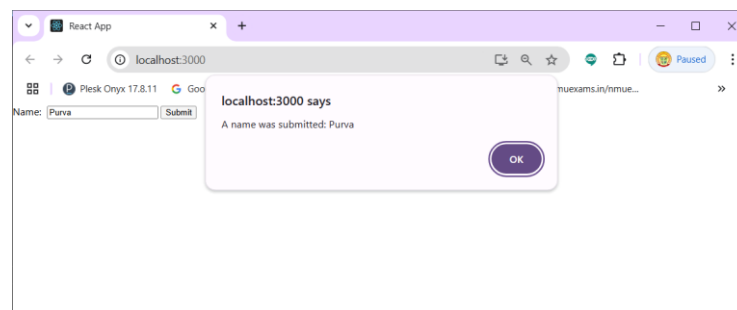
**//using React.createRef() method:**

```
import React from 'react';

function App() {
  // Step 1: Create refs for the input fields
  let inputRef = React.createRef();
  // Step 2: Handle form submission
  const handleSubmit = event => {
    alert('A name was submitted: ' + inputRef.current.value);
    event.preventDefault();
  };
  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name: &nbsp;
        <input type="text" ref={inputRef} />
      </label>
      &nbsp;
      <input type="submit" value="Submit" />
    </form>
  );
}

export default App;
```

**Output:**



**// Write ReactJs code for Applying form components using Uncontrolled Form**

**//using using useRef hook:**

```
import React, { useRef } from 'react';

const UncontrolledForm = () => {
  // Step 1: Create refs for the input fields
  const nameRef = useRef(null);

  // Step 2: Handle form submission
  const handleSubmit = (e) => {
    e.preventDefault();

    // Access the input values using refs
    const name = nameRef.current.value;
    alert('A name was submitted: ' + name);

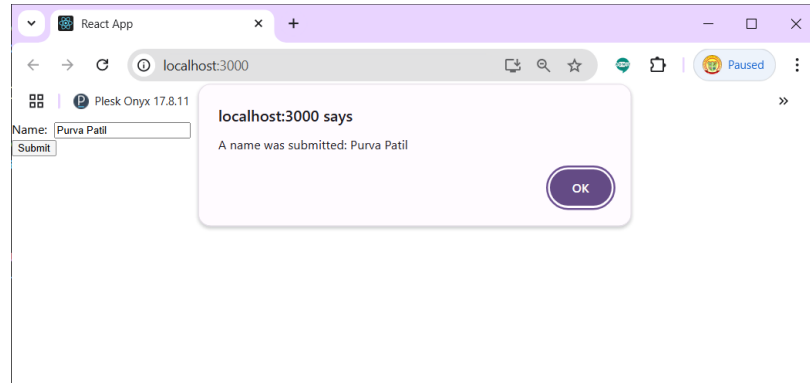
    // You can reset the form fields if needed
    nameRef.current.value = '';
  };

  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label>
          Name: &nbsp;
          <input
            type="text"
            ref={nameRef} // Attach the ref to the input
          />
        </label>
      </div>
      <button type="submit">Submit</button>
    </form>
  );
};

export default UncontrolledForm;
```



## Output:



## Assignments – 4 Write ReactJs code to create student Registration Form with Validation.

### App.js

```
import React, { useState } from "react";
import "./App.css";
function App() {
  const [firstName, setFirstName] = useState("");
  const [lastName, setLastName] = useState("");
  const [email, setEmail] = useState("");
  const [contact, setContact] = useState("");
  const [gender, setGender] = useState("male");
  const [subjects, setSubjects] = useState({
    computer: true,
    maths: false,
    physics: false,
  });
  const [selectedOption, setSelectedOption] = useState("");
  const [about, setAbout] = useState("");
  const [errors, setErrors] = useState({});
  const validate = () => {
    const errors = {};
    if (!firstName) errors.firstName = "First Name is required";
    if (!lastName) errors.lastName = "Last Name is required";
    if (!email) errors.email = "Email is required";
    else if (!/\S+@\S+\.\S+/.test(email)) errors.email = "Email address is
invalid";
    if (!contact) errors.contact = "Contact number is required";
    else if (!/^d{10}$/.test(contact)) errors.contact = "Contact number must be
10 digits";
    if (!about) errors.about = "About section is required";
    if (!selectedOption) errors.selectedOption = "Please select an option";
    if (!subjects.computer && !subjects.maths && !subjects.physics) {
      errors.subjects = "At least one subject must be selected";
    }
    return errors;
  }
}
```

```

};

const handleSubmit = (e) => {
  e.preventDefault();
  const validationErrors = validate();
  if (Object.keys(validationErrors).length > 0) {
    setErrors(validationErrors);
  } else {
    alert(
      `Student Details:
      First Name: ${firstName}
      Last Name: ${lastName}
      Email: ${email}
      Contact: ${contact}
      Gender: ${gender}
      Selected Option: ${selectedOption}
      Subjects: ${JSON.stringify(subjects)}
      About: ${about}`
    );
    // Add your form submission logic here
    setErrors({});
  }
};

const handleSubjectChange = (sub) => {
  setSubjects((prev) => ({
    ...prev,
    [sub]: !prev[sub],
  }));
};

const handleReset = () => {
  // Reset all state variables here
  setFirstName("");
  setLastName("");
  setEmail("");
  setContact("");
  setGender("male");

```

```

setSubjects({
  computer: true,
  maths: false,
  physics: false,
});
setSelectedOption("");
setAbout("");
setErrors({});
};
return (
  <div className="App">
    <h1>Form in React</h1>
    <fieldset>
      <form onSubmit={handleSubmit}>
        <label htmlFor="firstname">
          First Name*
          <input
            type="text"
            name="firstname"
            id="firstname"
            value={firstName}
            onChange={(e) => setFirstName(e.target.value)}
            placeholder="Enter First Name"
            required
            />
            {errors.firstName} && <span
className="error">{errors.firstName}</span>
          </label>
          <label htmlFor="lastname">
            Last Name*
            <input
              type="text"
              name="lastname"
              id="lastname"
              value={lastName}

```

```

        onChange={(e) => setLastName(e.target.value)}
        placeholder="Enter Last Name"
        required
      />
      {errors.lastName && <span
className="error">{errors.lastName}</span>}
    </label>
    <label htmlFor="email">
      Enter Email*
      <input
        type="email"
        name="email"
        id="email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        placeholder="Enter email"
        required
      />
      {errors.email && <span className="error">{errors.email}</span>}
    </label>
    <label htmlFor="tel">
      Contact*
      <input
        type="tel"
        name="contact"
        id="contact"
        value={contact}
        onChange={(e) => setContact(e.target.value)}
        placeholder="Enter Mobile number"
        required
      />
      {errors.contact && <span
className="error">{errors.contact}</span>}
    </label>
    <label htmlFor="gender">

```



Gender\*

```
<input
  type="radio"
  name="gender"
  value="male"
  id="male"
  checked={gender === "male"}
  onChange={(e) => setGender(e.target.value)}
/>
```

Male

```
<input
  type="radio"
  name="gender"
  value="female"
  id="female"
  checked={gender === "female"}
  onChange={(e) => setGender(e.target.value)}
/>
```

Female

```
<input
  type="radio"
  name="gender"
  value="other"
  id="other"
  checked={gender === "other"}
  onChange={(e) => setGender(e.target.value)}
/>
```

Other

</label>

<label htmlFor="lang">Your best Subject</label>

```
<input
  type="checkbox"
  name="lang"
  id="computer"
  checked={subjects.computer === true}
```

```

        onChange={(e) => handleSubjectChange("computer")}}
      />
      Computer Science
      <input
        type="checkbox"
        name="lang"
        id="maths"
        checked={subjects.maths === true}
        onChange={(e) => handleSubjectChange("maths")}}
      />
      Maths
      <input
        type="checkbox"
        name="lang"
        id="physics"
        checked={subjects.physics === true}
        onChange={(e) => handleSubjectChange("physics")}}
      />
      Physics
      <br></br>
      {errors.subjects} && <span
className="error">{errors.subjects}</span>
      <label>Select your choice</label>
      <select
        name="select"
        id="select"
        value={selectedOption}
        onChange={(e) => setSelectedOption(e.target.value)}
      >
        <option value="" disabled selected={selectedOption === ""}>
          Select your Ans
        </option>
        <optgroup label="Beginers">
          <option value="1">HTML</option>
          <option value="2">CSS</option>

```

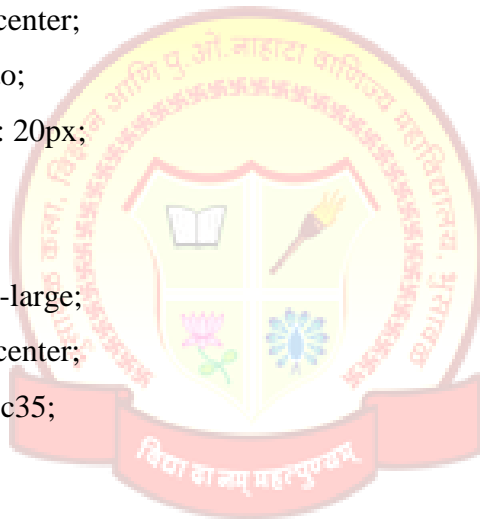
```

      <option value="3">JavaScript</option>
    </optgroup>
    <optgroup label="Advance">
      <option value="4">React</option>
      <option value="5">Node</option>
      <option value="6">Express</option>
      <option value="7">MongoDB</option>
    </optgroup>
  </select>
  {errors.selectedOption && (
    <span className="error">{errors.selectedOption}</span>
  )}
  <label htmlFor="about">About</label>
  <textarea
    name="about"
    id="about"
    cols="30"
    rows="10"
    onChange={(e) => setAbout(e.target.value)}
    placeholder="About yourself"
    value={about}
    required
  ></textarea>
  {errors.about && <span className="error">{errors.about}</span>}
  <button type="reset" onClick={() => handleReset()}>
    Reset
  </button>
  <button type="submit">Submit</button>
</form>
</fieldset>
</div>
);
}
export default App;

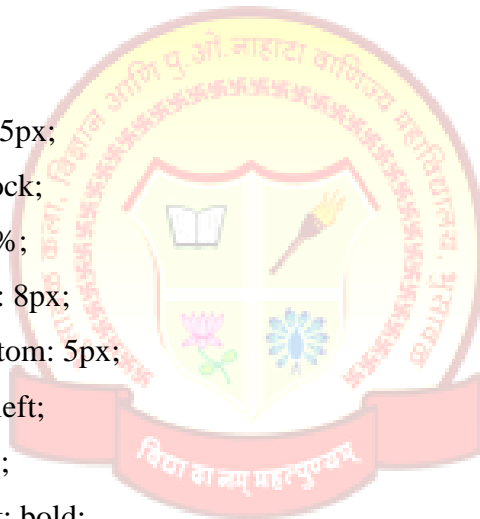
```

## App.css

```
/* Filename - App.css */
body {
    background: #f3f3f3;
    text-align: center;
}
.App {
    background-color: #fff;
    border-radius: 15px;
    box-shadow: 0 0 20px rgba(0, 0, 0, 0.2);
    padding: 10px 20px;
    transition: transform 0.2s;
    width: 500px;
    text-align: center;
    margin: auto;
    margin-top: 20px;
}
h1 {
    font-size: x-large;
    text-align: center;
    color: #327c35;
}
fieldset {
    border: none;
}
input {
    display: block;
    width: 100%;
    /* margin-bottom: 15px; */
    padding: 8px;
    box-sizing: border-box;
    border: 1px solid #ddd;
    border-radius: 3px;
    font-size: 12px;
}
```



```
input[type="radio"],
input[type="checkbox"] {
    display: inline;
    width: 10%;
}
select {
    display: block;
    width: 100%;
    margin-bottom: 15px;
    padding: 10px;
    box-sizing: border-box;
    border: 1px solid #ddd;
    border-radius: 5px;
}
label {
    font-size: 15px;
    display: block;
    width: 100%;
    margin-top: 8px;
    margin-bottom: 5px;
    text-align: left;
    color: #555;
    font-weight: bold;
}
button {
    padding: 15px;
    border-radius: 10px;
    margin: 15px;
    border: none;
    color: white;
    cursor: pointer;
    background-color: #4caf50;
    width: 40%;
    font-size: 16px;
}
```



```

textarea {
  resize: none;
  width: 98%;
  min-height: 100px;
  max-height: 150px;
}

.error {
  color: red;
  font-size: 0.875em;
  margin-top: 0.25em;
}

```

## Output:

Form in React

First Name\*  
Purna

Last Name\*  
Juni

Enter Email\*  
juni@  
Please include an @ in the email address. 'juni' is missing an @

Contact\*  
Enter Mobile number

Gender\*  
☒ Male
 ☐ Female
 ☐ Other

Your best Subject  
☒ Computer Science
 ☐ Maths
 ☐ Physics

Select your choice  
 Select your Ans

About  
 about: your text

Reset Submit

Form in React

First Name\*  
Purna

Last Name\*  
Juni

Enter Email\*  
juni@

Gender\*  
☒ Male
 ☐ Female
 ☐ Other

Your best Subject  
☒ Computer Science
 ☐ Maths
 ☐ Physics

Select your choice  
 Select your Ans

About  
 about: your text

Reset Submit

React App localhost:3000

### Form in React

First Name\*  
Punya

Last Name\*  
Patil

Enter Email\*  
punvaspatil@gmail.com

Contact\*  
9975114205

Gender\* ☒ Male ☐ Female ☐ Other

Your best Subject  
☐ Computer Science ☐ Maths ☐ Physics  
At least one subject must be selected

Select your choice  
Select your Ans ▼

Please select an option

About  
Hello TYRCA

Reset Submit

React App localhost:3000

### Form in React

First Name\*  
Punya

Last Name\*  
Patil

Enter Email\*  
punvaspatil@gmail.com

Contact\*  
9975114205

Gender\* ☒ Male ☐ Female ☐ Other

Your best Subject  
☒ Computer Science ☐ Maths ☐ Physics

Select your choice  
React ▼

About  
Hello TYRCA

Reset Submit

React App localhost:3000

**localhost:3000 says**

Student Details:  
First Name: Punya  
Last Name: Patil  
Email: punvaspatil@gmail.com  
Contact: 9975114205  
Gender: male  
Selected Option: 4  
Subjects: ["computer":true,"maths":false,"physics":false]  
About: Hello TYRCA

OK

Gender\* ☒ Male ☐ Female ☐ Other

Your best Subject  
☒ Computer Science ☐ Maths ☐ Physics

Select your choice  
React ▼

About  
Hello TYRCA

Reset Submit

## Assignments – 5 Write ReactJs code to create Simple Login Form.

### App.JS:

```
import React, { useState } from 'react';
import './App.css'; // Importing the CSS file
const LoginForm = () => {
  const [formData, setFormData] = useState({
    username: "",
    password: "",
  });
  const [errors, setErrors] = useState({});
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({
      ...formData,
      [name]: value,
    });
  };
  const validate = () => {
    const newErrors = {};
    if (!formData.username) {
      newErrors.username = 'Username is required';
    }
    if (!formData.password) {
      newErrors.password = 'Password is required';
    }
    return newErrors;
  };
  const handleSubmit = (e) => {
    e.preventDefault();
    const validationErrors = validate();
    if (Object.keys(validationErrors).length === 0) {
      // No errors, submit the form
      alert('Login successful!');
      console.log(formData);
      setFormData({ username: "", password: "" }); // Reset form
    }
  };
}
```



```

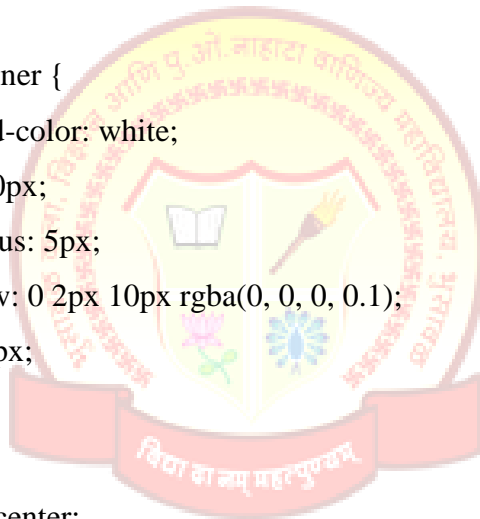
    } else {
      setErrors(validationErrors);
    }
  };
  return (
    <div className="login-container">
      <h2>Login Form</h2>
      <form onSubmit={handleSubmit} className="login-form">
        <div className="form-group">
          <label>Username:</label>
          <input
            type="text"
            name="username"
            value={formData.username}
            onChange={handleChange}
            className={errors.username ? 'error' : ''}
          />
          {errors.username && <span className="error-
message">{errors.username}</span>}
        </div>
        <div className="form-group">
          <label>Password:</label>
          <input
            type="password"
            name="password"
            value={formData.password}
            onChange={handleChange}
            className={errors.password ? 'error' : ''}
          />
          {errors.password && <span className="error-
message">{errors.password}</span>}
        </div>
        <button type="submit" className="submit-button">Login</button>
      </form>
    </div>
  );

```

```
);  
};  
export default LoginForm;
```

### App.CSS:

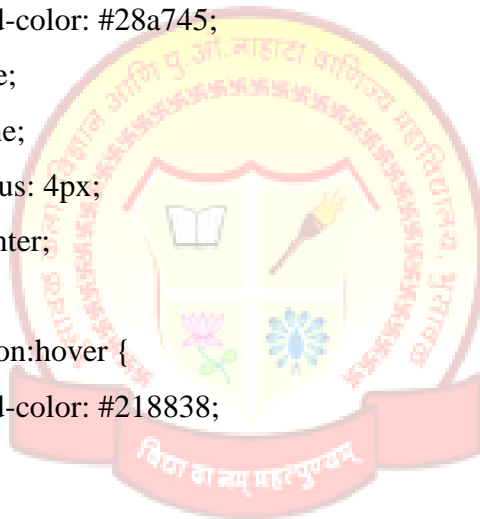
```
body {  
  font-family: Arial, sans-serif;  
  background-color: #f4f4f4;  
  display: flex;  
  justify-content: center;  
  align-items: center;  
  height: 100vh;  
  margin: 0;  
}  
.login-container {  
  background-color: white;  
  padding: 20px;  
  border-radius: 5px;  
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);  
  width: 300px;  
}  
h2 {  
  text-align: center;  
  margin-bottom: 20px;  
}  
.form-group {  
  margin-bottom: 15px;  
}  
label {  
  display: block;  
  margin-bottom: 5px;  
}  
input {  
  width: 100%;  
  padding: 8px;
```



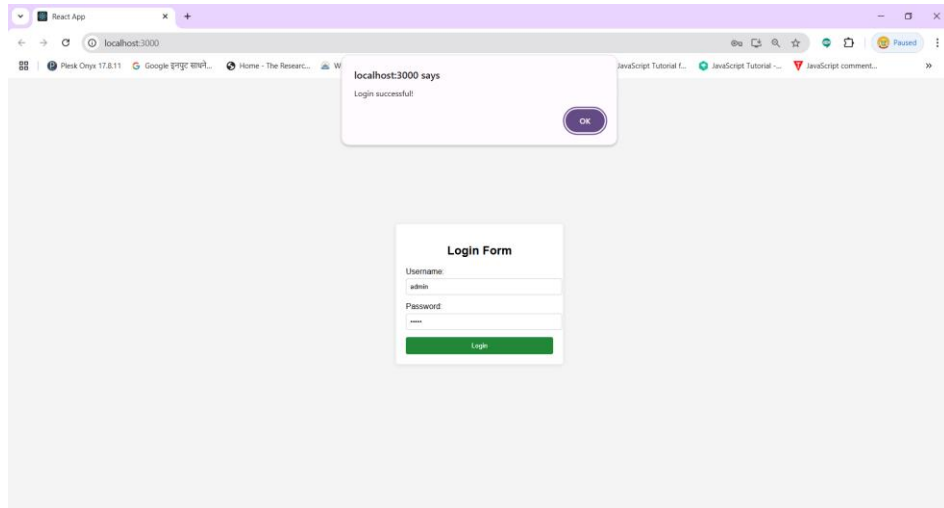
```

border: 1px solid #ccc;
border-radius: 4px;
}
input.error {
border-color: red;
}
.error-message {
color: red;
font-size: 12px;
}
.submit-button {
width: 100%;
padding: 10px;
background-color: #28a745;
color: white;
border: none;
border-radius: 4px;
cursor: pointer;
}
.submit-button:hover {
background-color: #218838;
}

```



## Output:



## Assignments – 6 Write ReactJs Create a Single Page Application.

### // Create a Single Page Application Using Routing

#### Step 1: Set Up Your React Project

Create React App:

```
npx create-react-app my-routing-app
```

```
cd my-routing-app
```

#### Step 2: Install React Router

Need to install `react-router-dom`, which is the library for routing in React:

```
npm install react-router-dom
```

#### Step 3: Create Your Components

Create the following components: `Home`, `About`, `Contact`, `Team`, and `Company`. Each component will represent a different page in our application.

##### 1. Home.js

```
// src/Home.js
```

```
import React from "react";
import { useNavigate } from "react-router-dom";
const Home = () => {
  const navigate = useNavigate();
  return (
    <div>
      <h2>Home Page</h2>
      <button onClick={() => navigate("/contact")}>Go to
Contact</button>
    </div>
  );
};
export default Home;
```

##### 2. About.js

```
// src/About.js
```

```
import React from "react";
import { Link, Outlet } from "react-router-dom";
const About = () => {
  return (
    <div>
      <h2>About Page</h2>
```

```

    <nav>
      <ul>
        <li>
          <Link to="team">Our Team</Link>
        </li>
        <li>
          <Link to="company">Our Company</Link>
        </li>
      </ul>
    </nav>
    <Outlet />
  </div>
);
};
export default About;

```

### 3. Contact.js

```

// src/Contact.js
import React from "react";
const Contact = () => {
  return <h2>Contact Page</h2>;
};
export default Contact;

```

### 4. Team.js

```

// src/Team.js
import React from "react";
const Team = () => {
  return <h2>Team Page</h2>;
};
export default Team;

```

### 5. Company.js

```

// src/Company.js
import React from "react";
const Company = () => {
  return <h2>Company Page</h2>;
};

```

```
export default Company;
```

#### Step 4: Set Up Routing in App.js

Create the main application file (`App.js`) that incorporates all these components and sets up routing.

```
// src/App.js
```

```
import React from "react";
```

```
import {
```

```
  BrowserRouter as Router,
```

```
  Routes,
```

```
  Route,
```

```
  Link,
```

```
} from "react-router-dom";
```

```
import Home from "./Home";
```

```
import About from "./About";
```

```
import Contact from "./Contact";
```

```
import Team from "./Team";
```

```
import Company from "./Company";
```

```
function App() {
```

```
  return (
```

```
    <Router>
```

```
      <nav>
```

```
        <ul>
```

```
          <li>
```

```
            <Link to="/">Home</Link>
```

```
          </li>
```

```
          <li>
```

```
            <Link to="/about">About</Link>
```

```
          </li>
```

```
          <li>
```

```
            <Link to="/contact">Contact</Link>
```

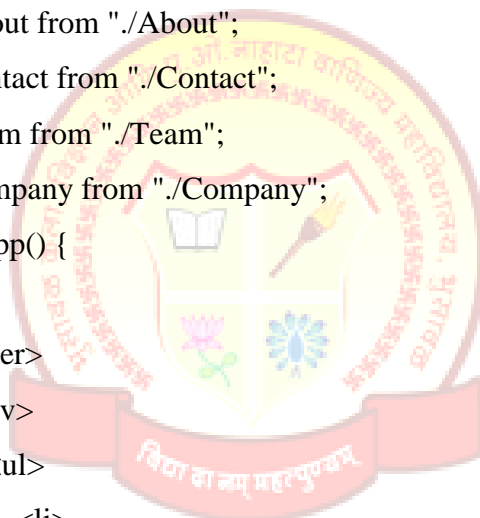
```
          </li>
```

```
        </ul>
```

```
      </nav>
```

```
      { /* Implementing Routes for respective Path */ }
```

```
      <Routes>
```



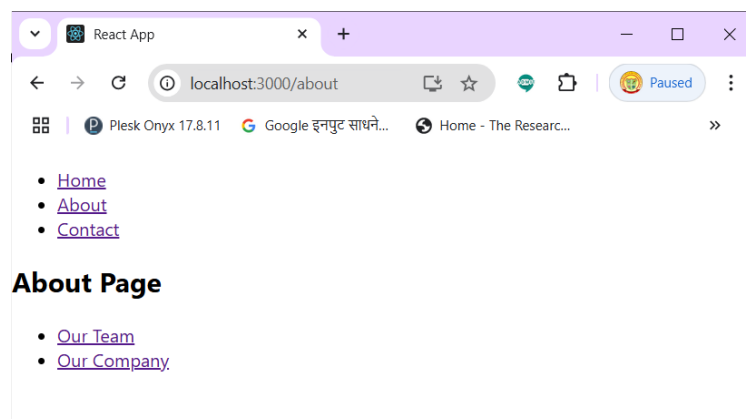
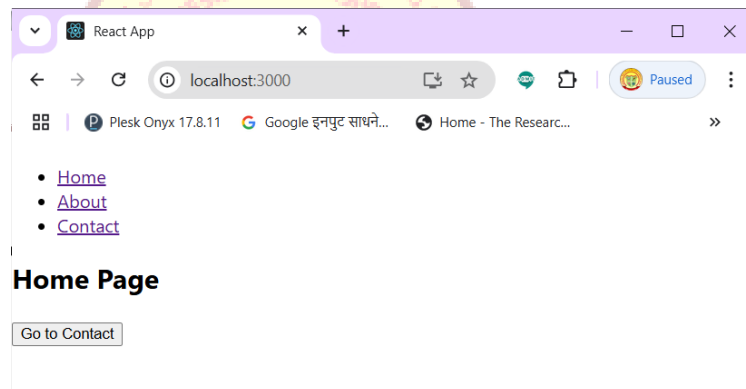
```

<Route path="/" element={<Home />} />
<Route path="/about" element={<About />} />
  <Route path="team" element={<Team />} />
  <Route path="company" element={<Company />} />
</Route>
<Route path="/contact" element={<Contact />} />
</Routes>
</Router>
);
}
export default App;

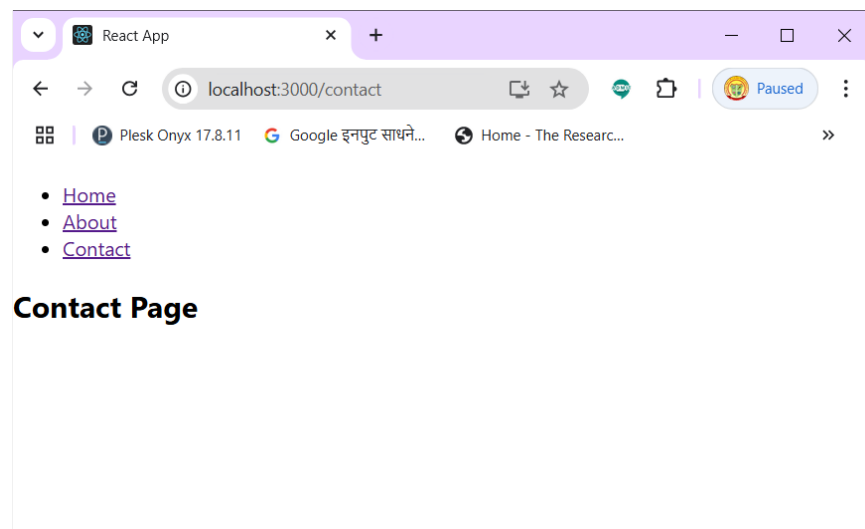
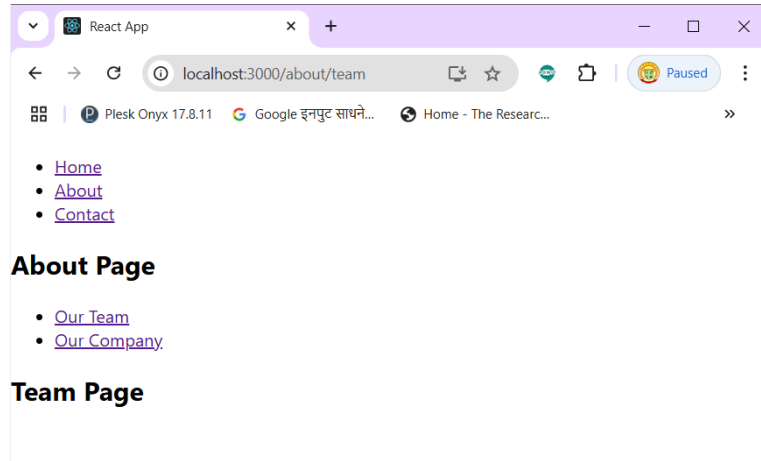
```

## Step 5: Run Your Application

**npm start**







## // Create a College Website using Single Page Application (Routing)

### Step - 1. Set Up the React Project

First, set up a new React project using Create React App.

- `npx create-react-app college-website`
- `cd college-website`

### Step - 2. Create the Project Structure

Create a `components` folder inside the `src` directory to organize our components.

```
college-website/  
+- - node_modules/  
+- - public/  
+- - src/  
| +- - components/  
| | +- - Header.js  
| | +- - Footer.js  
| | +- - Home.js  
| | +- - About.js  
| | +- - Courses.js  
| | +- - Contact.js  
| +- - App.css  
| +- - App.js  
| +- - index.js  
+- - package.json  
+- - README.md
```

### Step - 3. Create Individual Components

#### 1. Header Component

This component will contain the navigation menu.

**File: `src/components/Header.js`**

```
import React from 'react';
```

```
import { Link } from 'react-router-dom';
```

```
const Header = () => {
```

```
  return (
```

```
    <header>
```

```
    <h1> Bhusawal Arts, Science and P. O. Nahata Commerce
```

```
      College, Bhusawal </h1>
```

```
    <nav>
```

```
      <ul>
```

```
        <li><Link to="/">Home</Link></li>
```

```
        <li><Link to="/about">About</Link></li>
```

```
        <li><Link to="/courses">Courses</Link></li>
```

```
        <li><Link to="/contact">Contact</Link></li>
```

```
      </ul>
```

```
    </nav>
```

```
</header>

);

};

export default Header;
```

## 2. Footer Component

This component will display the footer of the webpage.

**File: `src/components/Footer.js`**

```
import React from 'react';

const Footer = () => {

  return (

    <footer>

    <p>&copy; 2024 Bhusawal Arts, Science and P. O. Nahata Commerce
    College, Bhusawal. All rights reserved.</p>

    </footer>

  );

};

export default Footer;
```

## 3. Home Component

This component will display the homepage content.

**File: `src/components/Home.js`**

```
import React from 'react';

const Home = () => {

  return (

    <div>

    <h2>Welcome to Bhusawal Arts, Science and P. O. Nahata Commerce
    College, Bhusawal </h2>

    <p>This is the home page of the college website.</p>

    </div>

  );

};

export default Home;
```

## 4. About Component

This component will display the about page content.

**File: `src/components/About.js`**

```
import React from 'react';
```

```
const About = () => {
```

```
  return (
```

```
    <div>
```

```
      <h2>About Us</h2>
```

<p> With a view to promote Quality in Higher Education in the rural area of Bhusawal Tahasil of Jalgaon District (earlier known as East Khandesh). Tapti Education Society has been promoted by social workers and Philanthropists way back in 1958. The Bhusawal Arts, Science and P. O. Nahata Commere College, Bhusawal was then established by Tapti Education Society in 1963, the first college in Bhusawal Tahasil.</p>

<p>Our college got accredited in 2001, 2008 and in 2014 consecatively and achived , A(CGPA 3.28) and A(CGPA 3.30) respectively; thus being the first college in North Maharashtra University Jalgaon jurisdiction to achive hattrick of 'A' Grade. </p>

```
    </div>
```

```
  );
```

```
};
```

```
export default About;
```

## 5. Courses Component

This component will display the courses page content.

**File: `src/components/Courses.js`**

```
import React from 'react';
```

```
const Courses = () => {
```

```
  return (
```

```
    <div>
```

```
      <h2>UG Courses</h2>
```

```
      <ul>
```

```
        <li>BSC</li>
```

```
        <li>BCA</li>
```

```
        <li>BVOC</li>
```

```
        <li>BA</li>
```

```
        <li>BCOM</li>
```

```
      </ul>
```

```
      <h2>PG Courses</h2>
```

```

    </ul>
    <li>MSC</li>
    <li>MA</li>
    <li>MCOM</li>
  </ul>
</div>

);
};

export default Courses;

```

## 6. Contact Component

This component will display the contact page content.

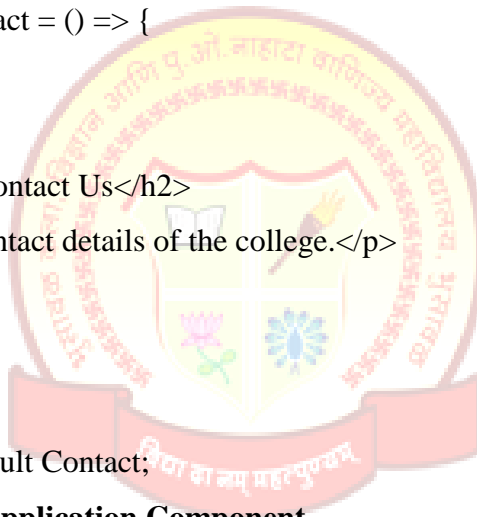
**File: `src/components/Contact.js`**

```

import React from 'react';
const Contact = () => {
  return (
    <div>
      <h2>Contact Us</h2>
      <p>Contact details of the college.</p>
    </div>
  );
};

export default Contact;

```



## Step - 4. Main Application Component

Now, we will use these components in the main `App.js` file and set up routing.

**File: `src/App.js`**

```

import React from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import Header from './components/Header';
import Footer from './components/Footer';
import Home from './components/Home';
import About from './components/About';
import Courses from './components/Courses';
import Contact from './components/Contact';
import './App.css';

```

```

const App = () => {
  return (
    <Router>
      <div className="app">
        <Header />
        <main>
          <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/about" element={<About />} />
            <Route path="/courses" element={<Courses />} />
            <Route path="/contact" element={<Contact />} />
          </Routes>
        </main>
        <Footer />
      </div>
    </Router>
  );
};

export default App;

```

### Step - 5. Styling the Components

Add some basic styles to make the UI look better.

**File: `src/App.css`**

```

body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
  background-color: #f0f2f5;
}

header {
  background-color: #333;
  color: white;
  padding: 10px 0;
  text-align: center;
}

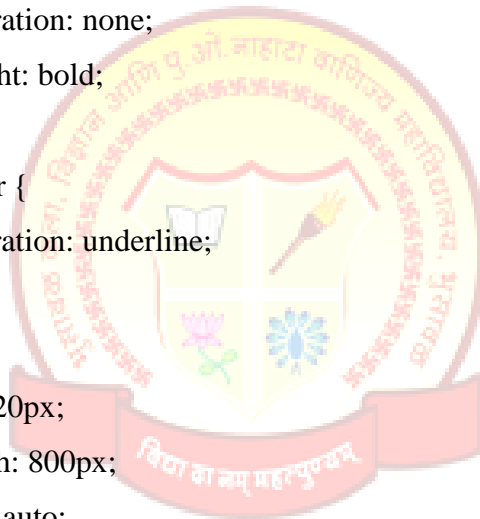
header h1 {

```

```

margin: 0;
}
nav ul {
list-style: none;
padding: 0;
display: flex;
justify-content: center;
}
nav li {
margin: 0 15px;
}
nav a {
color: white;
text-decoration: none;
font-weight: bold;
}
nav a:hover {
text-decoration: underline;
}
main {
padding: 20px;
max-width: 800px;
margin: 0 auto;
background-color: white;
border-radius: 8px;
box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
footer {
background-color: #333;
color: white;
text-align: center;
padding: 10px 0;
position: fixed;
width: 100%;
bottom: 0;

```



}

## Step - 6. Run Your Application

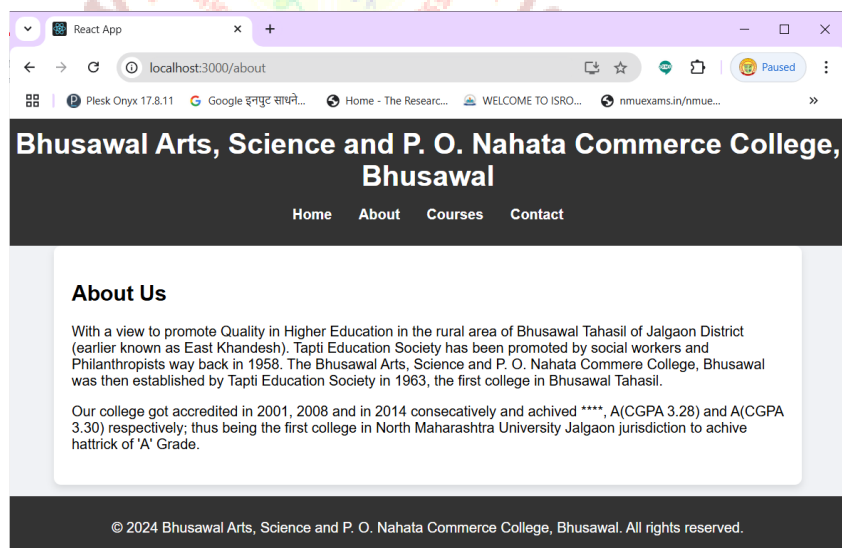
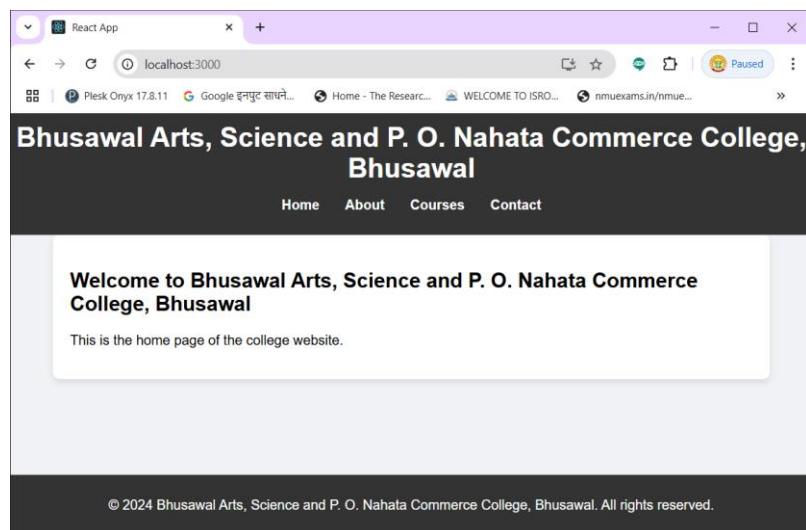
1. Install React Router:

**npm install react-router-dom**

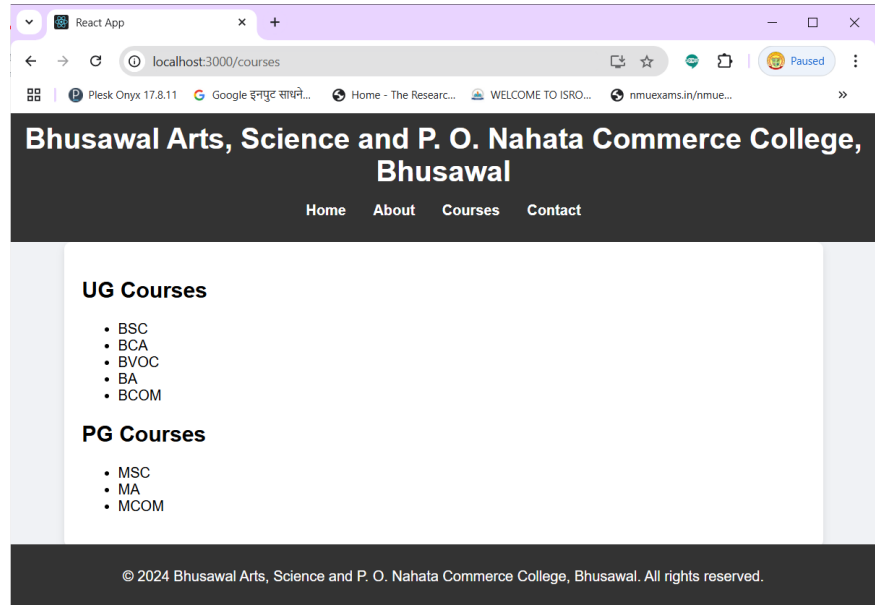
2. In the terminal, start the development server:

**npm start**

3. Open your browser and navigate to `http://localhost:3000`.







## Assignments – 7 Write NodeJs code to Applying Routing.

### Step by step routing example in Express.js:

#### Step 1: Set Up Your Express Application

- Create a new directory for your project:  
**mkdir express-routing-example**  
**cd express-routing-example**
- Initialize a new Node.js project:  
**npm init -y**
- Install Express.js:  
**npm install express**
- Create your main application file: Create a file named **app.js** in your project directory:

#### Step 2: Set Up Basic Express Server

- Open app.js in your code editor and add the following code to set up a basic Express server:

```
const express = require('express');  
const app = express();  
const port = 3000;
```

```
// Middleware to parse JSON bodies  
app.use(express.json());
```

```
// Define a route for the home page (GET request)  
app.get('/', (req, res) => {  
  res.send('Welcome to the Express Routing Example!');  
});
```

```
// Define a route for /about (GET request)  
app.get('/about', (req, res) => {  
  res.send('This is the about page.');
```

```
// Define a route for /contact (GET request)  
app.get('/contact', (req, res) => {  
  res.send('This is the contact page.');
```

```

});

// Define a route for /users/:id (GET request)
app.get('/users/:id', (req, res) => {
  const userId = req.params.id; // Get the user ID from the
  URL
  res.send(`User ID: ${userId}`);
});

// Define a route for creating a new user (POST request)
app.post('/users', (req, res) => {
  const newUser = req.body; // Get the user data from the
  request body
  // Logic to save the new user would go here
  res.status(201).send(`User created:
  ${JSON.stringify(newUser)}`);
});

// Handle 404 errors
app.use((req, res) => {
  res.status(404).send('404: Page Not Found');
});

// Start the server
app.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}`);
});

```

### Step 3: Run Your Express Application

Go back to your terminal and run the following command to start your Express server:

**node app.js**

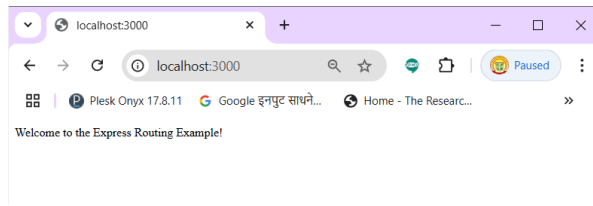
You should see a message indicating that the server is running:

**Server is running at http://localhost:3000**

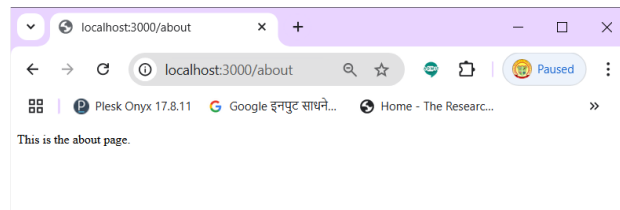
### Step 4: Test Your Application

Open a web browser and navigate to <http://localhost:3000>.

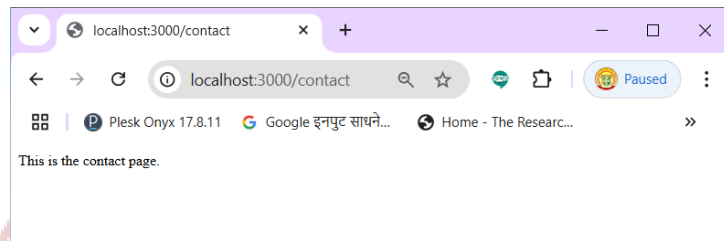
http://localhost:3000/ - Home Page



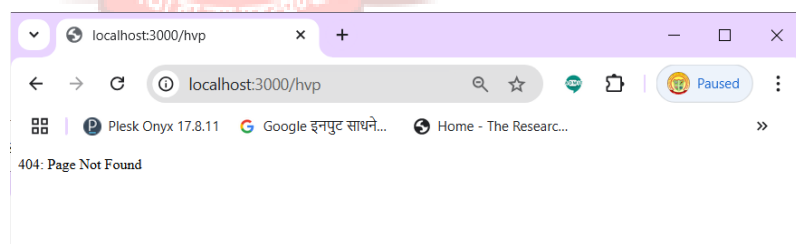
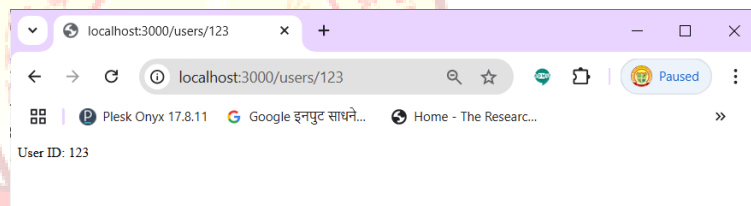
http://localhost:3000/about - About Page



http://localhost:3000/contact - Contact Page



http://localhost:3000/user/123 - User with ID 123



## Assignments – 8 Write ReactJs / NodeJs code to demonstrate the use of POST Method.

To demonstrate the use of the POST method using React.js for the frontend and Node.js (Express) for the backend, create a simple application where submit student data (name, age, city, class name, roll number) and send it to the server.

### Step 1: Setup the Backend with Node.js and Express

1. Initialize a new Node.js project:

```
mkdir student-app-backend  
cd student-app-backend  
npm init -y
```

2. Install necessary packages:

```
npm install express body-parser cors
```

3. Create the server file:

Create a file named `server.js`:

```
const express = require('express');  
const bodyParser = require('body-parser');  
const cors = require('cors');  
const app = express();  
const PORT = 5000;  
app.use(bodyParser.json());  
app.use(cors());  
app.post('/api/students', (req, res) => {  
  const student = req.body;  
  console.log('Received student data:', student);  
  res.status(201).json({ message: 'Student data received', student });  
});  
app.listen(PORT, () => {  
  console.log(`Server is running on http://localhost:${PORT}`);  
});
```

4. Start the server:

```
node server.js
```

### Step 2: Setup the Frontend with React

1. Initialize a new React project:

```
npx create-react-app student-app-frontend
```

## **cd student-app-frontend**

2. Install Axios for HTTP requests:

**npm install axios**

3. Create the StudentForm component:

Create a file named `StudentForm.js` in the `src` folder:

```
import React, { useState } from 'react';
```

```
import axios from 'axios';
```

```
import './StudentForm.css';
```

```
const StudentForm = () => {
```

```
  const [student, setStudent] = useState({
```

```
    name: "",
```

```
    age: "",
```

```
    city: "",
```

```
    className: "",
```

```
    rollNo: ""
```

```
  });
```

```
  const [message, setMessage] = useState("");
```

```
  const [messageType, setMessageType] = useState("");
```

```
  const handleChange = (e) => {
```

```
    const { name, value } = e.target;
```

```
    setStudent({ ...student, [name]: value });
```

```
  };
```

```
  const handleSubmit = (e) => {
```

```
    e.preventDefault();
```

```
    axios.post('http://localhost:5000/api/students', student)
```

```
      .then(response => {
```

```
        console.log(response.data);
```

```
        setMessage('Student data submitted successfully!');
```

```
        setMessageType('success');
```

```
        setStudent({
```

```
          name: "",
```

```
          age: "",
```

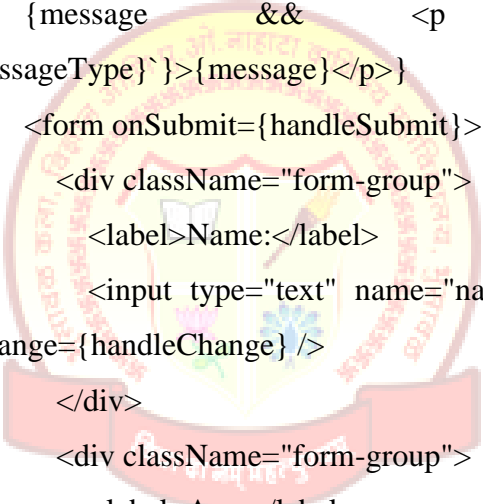
```
          city: "",
```

```
          className: "",
```

```

        rollNo: "
    });
})
.catch(error => {
    console.error('There was an error submitting the form!',
error);

    setMessage('Failed to submit student data.');
```



```

    setMessageType('error');
});

};

return (
    <div className="form-container">
        <h2>Student Form</h2>
        { message    &&    <p    className={`message
${messageType}` }>{message}</p>}
        <form onSubmit={handleSubmit}>
            <div className="form-group">
                <label>Name:</label>
                <input type="text" name="name" value={student.name}
onChange={handleChange} />
            </div>
            <div className="form-group">
                <label>Age:</label>
                <input type="number" name="age" value={student.age}
onChange={handleChange} />
            </div>
            <div className="form-group">
                <label>City:</label>
                <input type="text" name="city" value={student.city}
onChange={handleChange} />
            </div>
            <div className="form-group">
                <label>Class Name:</label>
                <input type="text" name="className"
value={student.className} onChange={handleChange} />

```

```

    </div>
    <div className="form-group">
      <label>Roll No:</label>
      <input type="text" name="rollNo" value={student.rollNo}
onChange={handleChange} />
    </div>
    <button type="submit">Submit</button>
  </form>
</div>
);
};
export default StudentForm;

```

#### 4. Add some basic CSS:

Create a file named `StudentForm.css` in the `src` folder:

```

.form-container {
  max-width: 40%;

  margin: 0 auto;
  padding: 20px;
  border: 1px solid #ccc;
  border-radius: 10px;
  background-color: #f9f9f9;
}

.form-container h2 {
  text-align: center;
  margin-bottom: 20px;
}

.message {
  text-align: center;
  margin-bottom: 20px;
  padding: 10px;
  border-radius: 5px;
}

.message.success {
  color: green;

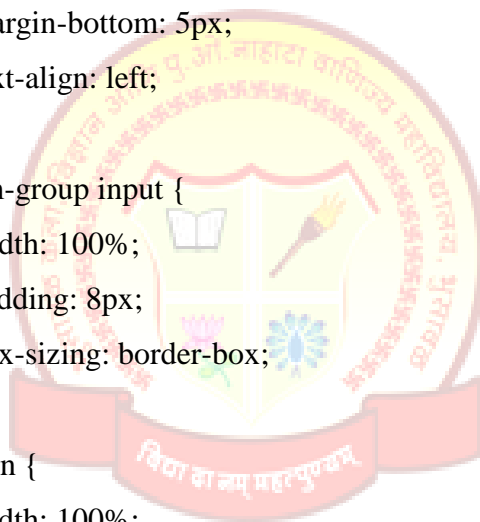
```



```

border: 1px solid green;
background-color: #e6ffe6;
}
.message.error {
color: red;
border: 1px solid red;
background-color: #ffe6e6;
}
.form-group {
margin-bottom: 15px;
}
.form-group label {
display: block;
margin-bottom: 5px;
text-align: left;
}
.form-group input {
width: 100%;
padding: 8px;
box-sizing: border-box;
}
button {
width: 100%;
padding: 10px;
background-color: #007bff;
color: #fff;
border: none;
border-radius: 5px;
cursor: pointer;
}
button:hover {
background-color: #0056b3;
}

```



##### 5. Modify the App component:

Update the `App.js` file to use the `StudentForm` component:

```

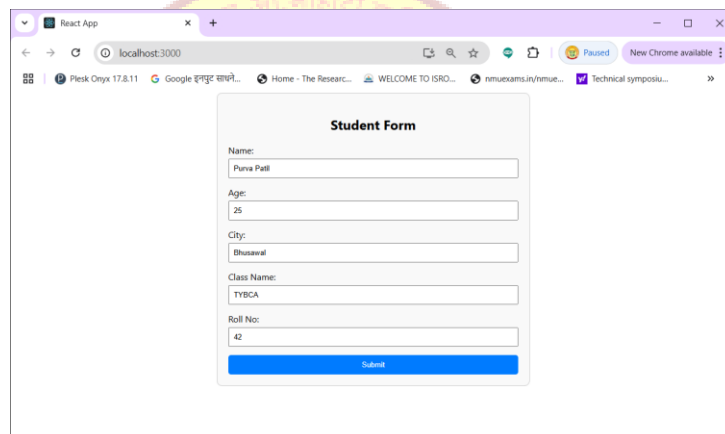
import React from 'react';
import './App.css';
import StudentForm from './StudentForm';
function App() {
  return (
    <div className="App">
      <StudentForm />
    </div>
  );
}
export default App;

```

6. Start the React application:

**npm start**

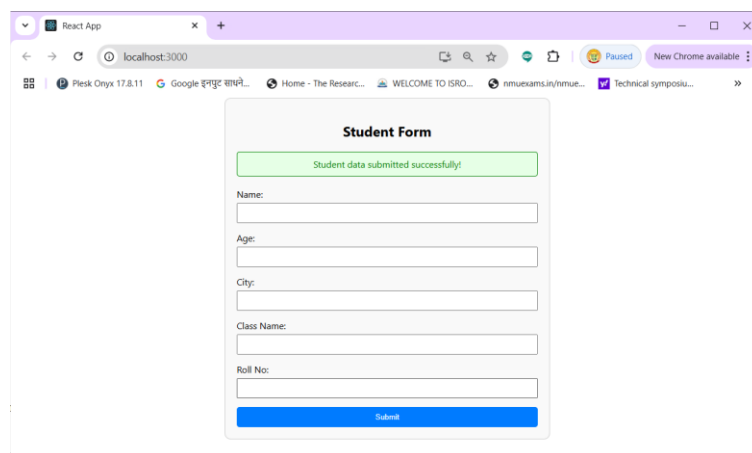
7. Open your browser and navigate to `http://localhost:3000`



The screenshot shows a web browser window with the title 'React App' and the address bar displaying 'localhost:3000'. The main content area features a 'Student Form' with the following fields and values:

- Name: Purna Patil
- Age: 25
- City: Bhosawal
- Class Name: TYBCA
- Roll No: 42

A blue 'Submit' button is located at the bottom of the form.



This screenshot shows the same web browser window after the form has been submitted. A green message box at the top of the form area displays the text: 'Student data submitted successfully!'. The input fields for Name, Age, City, Class Name, and Roll No are now empty. The blue 'Submit' button is still present at the bottom of the form.

```
C:\Windows\system32\cmd.exe - node server.js

D:\HVP-NodeJS-Examples\Practical-8\student-app-backend>node server.js
Server is running on http://localhost:5000
Received student data: {
  name: 'Purva Patil',
  age: '25',
  city: 'Bhusawal',
  className: 'TYBCA',
  rollNo: '42'
}
```



## Assignments – 9 Write ReactJs/ NodeJs code to demonstrate the use of GET Method.

To demonstrate the use of the GET method with React.js and Node.js, we can create a simple application where we fetch student data from a Node.js server using React.js.

### Step 1: Set Up the Node.js Backend

#### 1. Initialize the Node.js Project:

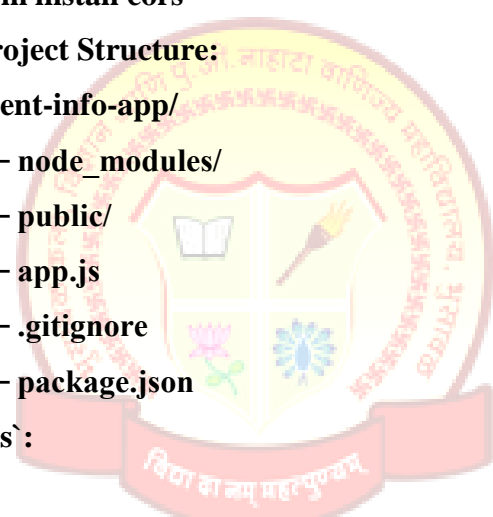
```
mkdir student-info-app
cd student-info-app
npm init -y
```

#### 2. Install Express:

```
npm install express
npm install cors
```

#### 3. Create the Project Structure:

```
student-info-app/
├── node_modules/
├── public/
├── app.js
├── .gitignore
└── package.json
```



#### 4. Create `app.js`:

```
// app.js
const express = require('express');
const cors = require('cors'); // Import cors
const app = express();
const port = 3000;
// Enable CORS
app.use(cors());
// Sample data
const students = [
  { id: 1, name: 'Purva', age: 20, city: 'Bhusawal', className: 'TYBCA',
    rollNo: 101 },
  { id: 2, name: 'Bhakti', age: 22, city: 'Bhusawal', className: 'TYBCA',
    rollNo: 102 },
```

```

    { id: 3, name: 'Malhar', age: 21, city: 'Varangaon', className: 'TYBCA',
    rollNo: 103 },
  ];
  // Route to get students data
  app.get('/students', (req, res) => {
    res.json(students);
  });
  // Start server
  app.listen(port, () => {
    console.log(`Server is running on http://localhost:${port}`);
  });

```

## Step 2: Set Up the React Frontend

### 1. Initialize the React Project:

```

npx create-react-app student-info-frontend
cd student-info-frontend

```

### 2. Install Axios (for making HTTP requests):

```

npm install axios

```

### 3. Create the Project Structure:

```

student-info-frontend/
├── node_modules/
├── public/
├── src/
│   ├── components/
│   │   └── StudentList.js
│   ├── App.js
│   └── index.js
├── .gitignore
├── package.json
└── README.md

```

### 4. Create `StudentList.js` in the `src/components` directory:

```

// src/components/StudentList.js
import React, { useEffect, useState } from 'react';
import axios from 'axios';

const StudentList = () => {

```

```

const [students, setStudents] = useState([]);

useEffect(() => {
  axios.get('http://localhost:3000/students')
    .then(response => {
      setStudents(response.data);
    })
    .catch(error => {
      console.error('There was an error fetching the data!', error);
    });
}, []);

return (
  <div>
    <h1>Student List</h1>
    <ul>
      {students.map(student => (
        <li key={student.id}>
          {student.name}, {student.age} years old, from
          {student.city}, enrolled in {student.className}, Roll No:
          {student.rollNo}
        </li>
      ))}
    </ul>
  </div>
);
};

export default StudentList;

```

## 5. Update `App.js` to include `StudentList`:

```

// src/App.js
import React from 'react';
import './App.css';
import StudentList from './components/StudentList';

```

```
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <StudentList />
      </header>
    </div>
  );
}
export default App;
```

6. Update CSS in `App.css` (optional):

```
/* src/App.css */
.App {
  text-align: center;
}
.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}
```

### Step 3: Run the Application

#### 1. Start the Node.js Server:

In the `student-info-app` directory, start the server:

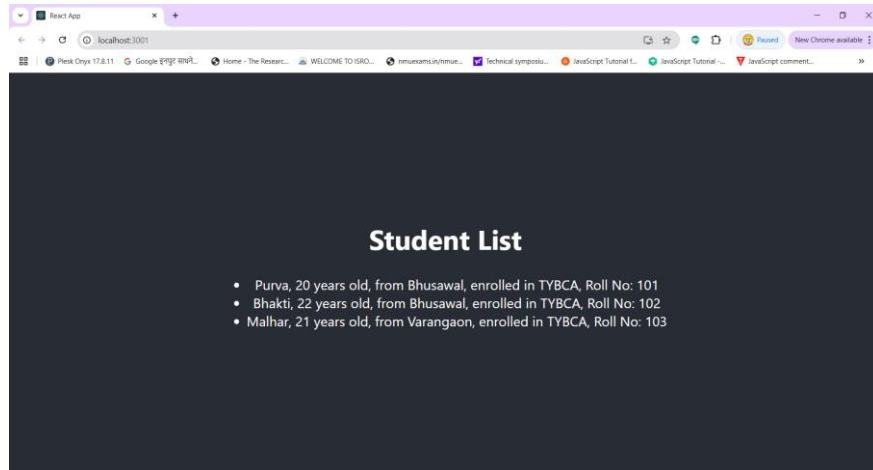
**node app.js**

#### 2. Start the React App:

In the `student-info-frontend` directory, start the React development server:

**npm start**

3. Open your browser and navigate to `http://localhost:3000` for the Node.js backend and `http://localhost:3000` for the React frontend.





## Assignments – 10 To demonstrate REST API in Node JS

**Create RESTful API for managing users application to handling HTTP requests and responses:**

This API will support the following operations:

- GET: Retrieve all users or a specific user by ID.
- POST: Create a new user.
- PUT: Update an existing user.
- DELETE: Delete a user.

### Step 1: Set Up Your Project

1. Create a new directory for your project:

```
mkdir user-httprequest-api
```

```
cd user-httprequest-api
```

2. Initialize a new Node.js project:

```
npm init -y
```

3. Install Express:

```
npm install express
```

4. Create the main application file: Create a file named **app.js**.

### Step 2: Implement the API in app.js file

```
const express = require('express');
const app = express();
const port = 3000;
// Middleware to parse JSON bodies
app.use(express.json());
// Sample data (in-memory storage)
let users = [
  { id: 1, name: 'Purva', email: 'purva5patil@gmail.com' },
  { id: 2, name: 'Bhakti', email: 'bhakti123@gmail.com' }
];
// GET request to retrieve all users
app.get('/users', (req, res) => {
  res.json(users);
});
// GET request to retrieve a user by ID
app.get('/users/:id', (req, res) => {
  const userId = parseInt(req.params.id);
```

```

const user = users.find(u => u.id === userId);
if (!user) {
  return res.status(404).json({ message: 'User not found' });
}
res.json(user);
});
// POST request to create a new user
app.post('/users', (req, res) => {
  const { name, email } = req.body;
  const newUser = {
    id: users.length + 1,
    name,
    email
  };
  users.push(newUser);
  res.status(201).json(newUser); // Respond with the created user
});
// PUT request to update a user by ID
app.put('/users/:id', (req, res) => {
  const userId = parseInt(req.params.id);
  const user = users.find(u => u.id === userId);
  if (!user) {
    return res.status(404).json({ message: 'User not found' });
  }
  const { name, email } = req.body;
  user.name = name || user.name; // Update name if provided
  user.email = email || user.email; // Update email if provided
  res.json(user); // Respond with the updated user
});
// DELETE request to delete a user by ID
app.delete('/users/:id', (req, res) => {
  const userId = parseInt(req.params.id);
  const userIndex = users.findIndex(u => u.id === userId);
  if (userIndex === -1) {
    return res.status(404).json({ message: 'User not found' });
  }

```

```

    }
    users.splice(userIndex, 1); // Remove the user from the array
    res.status(204).send(); // Respond with no content
  });
// Start the server
app.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}`);
});

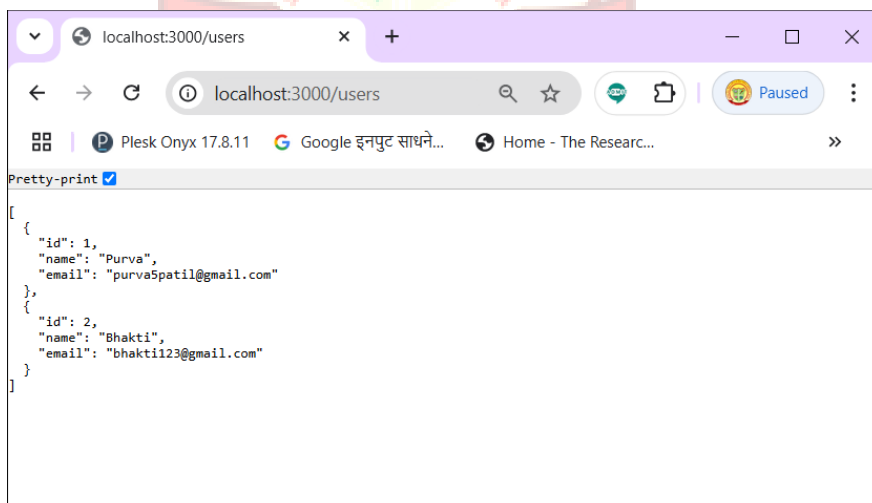
```

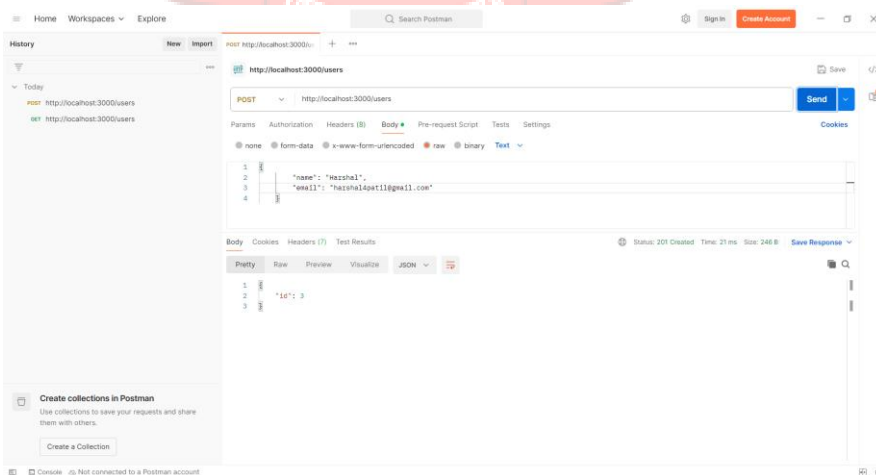
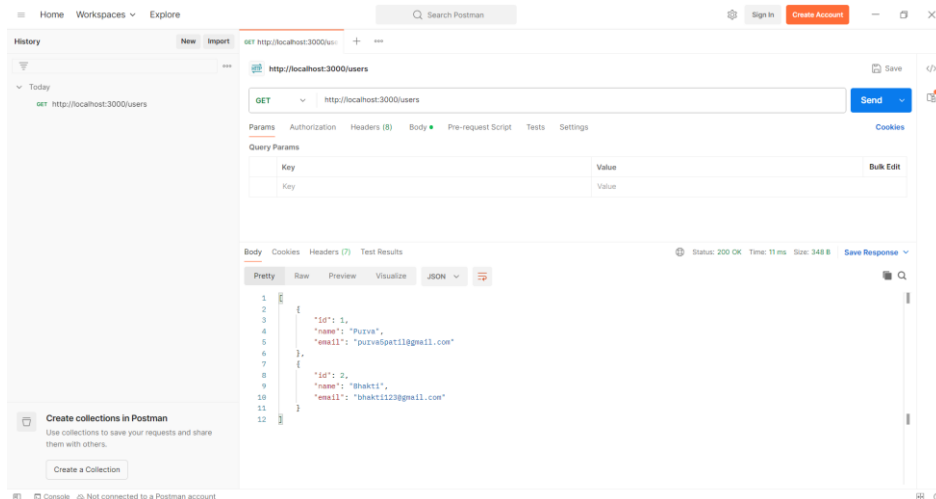
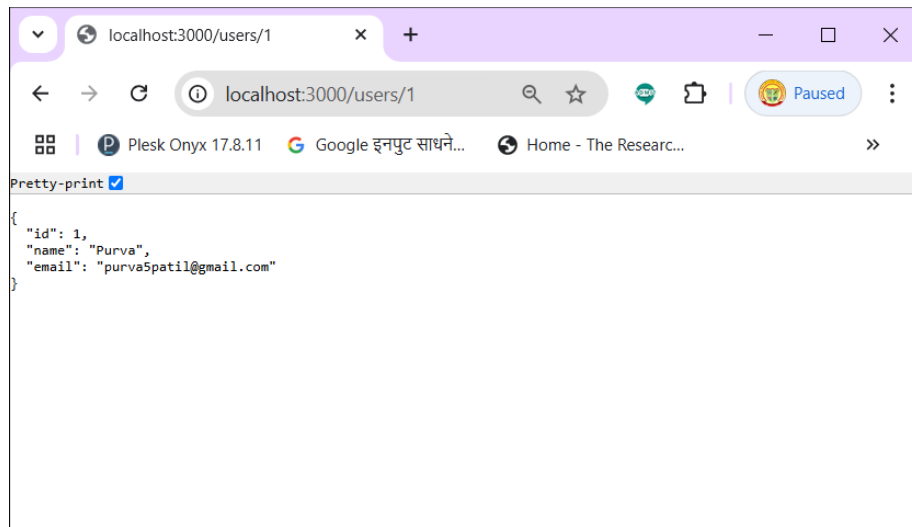
### Step 3: Run the Server

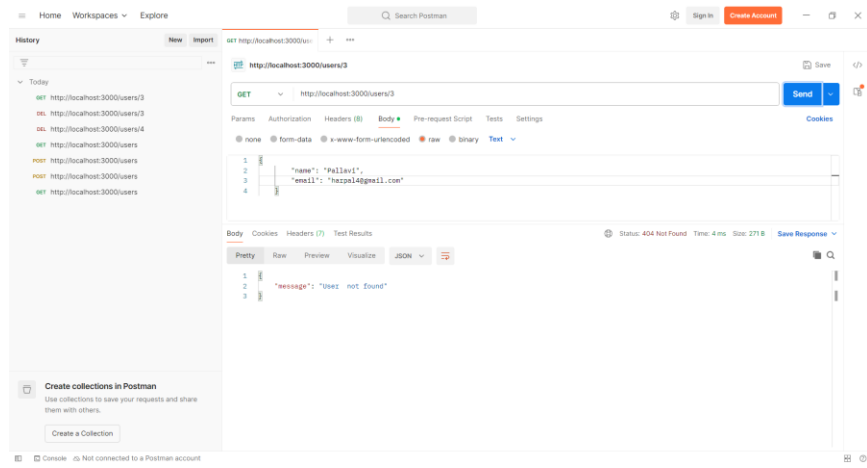
**node app.js**

**Step 4: Testing the API:** Use Postman or any API testing tool to test the following endpoints:

- Get all items (GET `http://localhost:3000/users`)
- Get a single item by ID (GET `http://localhost:3000/users/1`)
- Create a new item (POST `http://localhost:3000/users`)
  - Body: `{ "name": "User Name", "email": "user email" }`
- Update an existing item (PUT `http://localhost:3000/users/1`)
  - Body: `{ "name": "User Name", "email": "user email" }`
- Delete an item (DELETE `http://localhost:3000/users/1`)







## Assignments – 11 Create Node JS Application for to stored student information in database.

### Step 1: Set Up the Node.js Environment

#### 1. Initialize the Project:

```
mkdir pract-11
cd pract-11
npm init -y
```

#### 2. Install Dependencies:

```
npm install express mongoose body-parser ejs
```

### Step 2: Create the Project Structure

```
student-info-app/
├── node_modules/
├── public/
│   └── styles.css
├── views/
│   └── index.ejs
├── .gitignore
├── app.js
└── package.json
```

### Step 3: Set Up Mongoose and Connect to MongoDB

Create a file named `app.js` and set up the basic server and database connection:

```
// app.js
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const app = express();

// MongoDB connection
mongoose.connect('mongodb://localhost:27017/studentdb', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

const db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', () => {
```

```

    console.log('Connected to MongoDB');
  });
// Middleware
app.use(bodyParser.urlencoded({ extended: true }));
app.use(express.static('public'));
app.set('view engine', 'ejs');
// Student Schema
const studentSchema = new mongoose.Schema({
  name: String,
  age: Number,
  city: String,
  className: String,
  rollNo: Number,
});
const Student = mongoose.model('Student', studentSchema);
// Routes
app.get('/', (req, res) => {
  res.render('index');
});
app.post('/add-student', (req, res) => {
  const student = new Student(req.body);
  student.save((err) => {
    if (err) {
      console.log(err);
      res.send('Error saving student data');
    } else {
      res.render('index', { message: 'Student data successfully
inserted' });
    }
  });
});
// Start server
app.listen(3000, () => {
  console.log('Server is running on port 3000');
});

```

## Step 4: Create the Frontend

### 1. views/index.ejs:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Student Info</title>
  <link rel="stylesheet" href="/styles.css">
</head>
<body>
  <div class="container">
    <h1>Student Information</h1>
    <% if (typeof message !== 'undefined') { %>
      <p class="success"><%= message %></p>
    <% } %>
    <form action="/add-student" method="POST">
      <label for="name">Name:</label>
      <input type="text" id="name" name="name" required>

      <label for="age">Age:</label>
      <input type="number" id="age" name="age" required>

      <label for="city">City:</label>
      <input type="text" id="city" name="city" required>

      <label for="className">Class Name:</label>
      <input type="text" id="className" name="className"
required>

      <label for="rollNo">Roll No:</label>
      <input type="number" id="rollNo" name="rollNo" required>

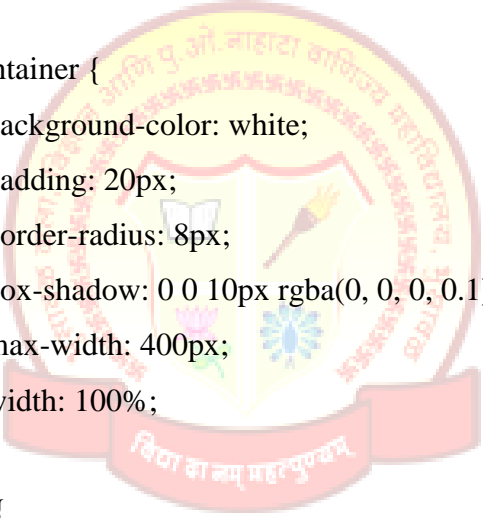
      <button type="submit">Submit</button>
```



```
</form>
</div>
</body>
</html>
```

## 2. public/styles.css:

```
body {
  font-family: Arial, sans-serif;
  background-color: #f4f4f4;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  margin: 0;
}
.container {
  background-color: white;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
  max-width: 400px;
  width: 100%;
}
h1 {
  margin-bottom: 20px;
}
label {
  display: block;
  margin-top: 10px;
}
input {
  width: 100%;
  padding: 8px;
  margin-top: 5px;
  border: 1px solid #ccc;
  border-radius: 4px;
}
```



```

}
button {
  display: block;
  width: 100%;
  padding: 10px;
  margin-top: 20px;
  background-color: #28a745;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}
button:hover {
  background-color: #218838;
}
.success {
  color: green;
  margin-bottom: 15px;
}

```

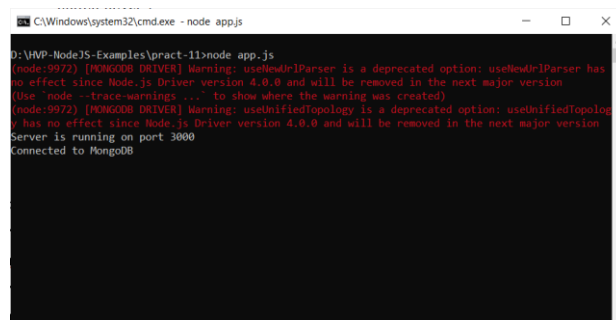
## Step 5: Run the Application

### 1. Start the MongoDB server:

```
mongod
```

### 2. Start the Node.js application:

```
node app.js
```



```

C:\Windows\system32\cmd.exe - node app.js
D:\VHP-NodeJS-Examples\pract-11>node app.js
(node:9972) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has
no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
(Use 'node --trace-warnings ...' to show where the warning was created)
(node:9972) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology
has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
Server is running on port 3000
Connected to MongoDB

```

### 3. Open your browser and navigate to `http://localhost:3000`.

Student Info

localhost:3000

Student Information

Name:  
Purva Harshal Patil

Age:  
23

City:  
Bhusawal

Class Name:  
TYBCA

Roll No:  
42

Submit

Student Info

localhost:3000/add-student

Student Information

Student data successfully inserted

Name:  
Purva Harshal Patil

Age:  
23

City:  
Bhusawal

Class Name:  
TYBCA

Roll No:  
42

Submit

mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

```
studentdb> db.student.find().pretty()

studentdb> db.students.find().pretty()
[
  {
    _id: ObjectId('67285cbbd3900938851ddf60'),
    name: 'Purva Harshal Patil',
    age: 23,
    city: 'Bhusawal',
    className: 'TYBCA',
    rollNo: 42,
    __v: 0
  }
]
studentdb> 
```

## Assignments – 12 Create Node JS Application for login credentials.

### Step 1: Set Up Your Project

1. Create a new directory for your project:

```
mkdir student-management-app
```

```
cd student-management-app
```

2. Initialize a new Node.js project:

```
npm init -y
```

3. Install the required packages:

```
npm install express mongoose bcryptjs express-session connect-mongo  
ejs
```

### Step 2: Create the Project Structure

**student-management-app**

```
├── models
│   ├── User.js
│   └── Student.js
├── routes
│   ├── auth.js
│   └── students.js
├── views
│   ├── index.ejs
│   ├── login.ejs
│   ├── home.ejs
│   └── students.ejs
├── public
│   └── styles.css
├── app.js
└── package.json
```

### Step 3: Create the Models

**`models/User.js` (for user authentication):**

```
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true }
```

```
});  
module.exports = mongoose.model('User', userSchema);
```

**`models/Student.js` (for student data):**

```
const mongoose = require('mongoose');  
const studentSchema = new mongoose.Schema({  
  name: { type: String, required: true },  
  age: { type: Number, required: true },  
  city: { type: String, required: true },  
  className: { type: String, required: true },  
  rollNo: { type: String, required: true, unique: true }  
});  
module.exports = mongoose.model('Student', studentSchema);
```

**Step 4: Set Up the Express Server**

**`app.js`:**

```
const express = require('express');  
const mongoose = require('mongoose');  
const session = require('express-session');  
const MongoStore = require('connect-mongo');  
const authRoutes = require('./routes/auth');  
const studentRoutes = require('./routes/students');  
const app = express();  
  
// Connect to MongoDB  
mongoose.connect('mongodb://localhost/student_management', {  
  useNewUrlParser: true,  
  useUnifiedTopology: true  
});  
  
// Middleware  
app.set('view engine', 'ejs');  
app.use(express.urlencoded({ extended: true }));  
app.use(express.static('public'));  
app.use(session({  
  secret: 'your-secret-key',
```

```

    resave: false,
    saveUninitialized: false,
    store: new MongoStore({
      mongoUrl: 'mongodb://localhost/student_management', // Ensure
correct MongoDB URI
      collectionName: 'sessions' // Optional
    })
  ));
// Define the home route
app.get('/', (req, res) => {
  res.render('index'); // Render the index.ejs view
});
// Routes
app.use('/auth', authRoutes);
app.use('/students', studentRoutes);
// Start server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});

```

### Step 5: Create Authentication Routes

**`routes/auth.js`:**

```

const express = require('express');
const bcrypt = require('bcryptjs');
const User = require('../models/User');
const router = express.Router();
const hashedPassword = bcrypt.hashSync('admin123', 8); // Hash the password
console.log(hashedPassword); // Use this hashed password in your MongoDB
insert command
// Login page
router.get('/login', (req, res) => {
  // Pass error variable if it exists, otherwise pass undefined
  res.render('login', { error: req.query.error || null });
});

```

```

// Handle login
router.post('/login', async (req, res) => {
  const { username, password } = req.body;
  const user = await User.findOne({ username });
  console.log(user); // Log the retrieved user object
  if (user && bcrypt.compareSync(password, user.password)) {
    req.session.userId = user._id;
    return res.redirect('/students');
  }
  res.render('login', { error: 'Invalid credentials' });
});

// Logout
router.get('/logout', (req, res) => {
  req.session.destroy(err => {
    if (err) return res.redirect('/students');
    res.redirect('/'); // Redirect to home page after logout
  });
});

// Register (optional)
router.get('/register', (req, res) => {
  res.render('register');
});

// Register a new user
router.post('/register', async (req, res) => {
  const { username, password } = req.body;
  const hashedPassword = bcrypt.hashSync(password, 8); // Hash the
password
  await User.create({ username, password: hashedPassword }); // Store the
user
  res.redirect('/login'); // Redirect to login page after registration
});
module.exports = router;

```

## Step 6: Create Student Management Routes

`routes/students.js`:`

```
const express = require('express');
const Student = require('../models/Student');
const router = express.Router();
// Middleware to check authentication
function isAuthenticated(req, res, next) {
  if (req.session.userId) {
    return next();
  }
  res.redirect('/login');
}
// Display all students
router.get('/', isAuthenticated, async (req, res) => {
  const students = await Student.find();
  res.render('students', { students });
});
// Add a new student
router.post('/add', isAuthenticated, async (req, res) => {
  await Student.create(req.body);
  res.redirect('/students');
});
// Edit a student
router.get('/edit/:id', isAuthenticated, async (req, res) => {
  const student = await Student.findById(req.params.id);
  res.render('edit', { student });
});
router.post('/edit/:id', isAuthenticated, async (req, res) => {
  await Student.findByIdAndUpdate(req.params.id, req.body);
  res.redirect('/students');
});
// Delete a student
router.get('/delete/:id', isAuthenticated, async (req, res) => {
  await Student.findByIdAndDelete(req.params.id);
  res.redirect('/students');
```



```
});  
module.exports = router;
```

## Step 7: Create Views

**`views/index.ejs`:**

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Student Management</title>  
  <link rel="stylesheet" href="/styles.css">  
</head>  
<body>  
  <div class="container">  
    <h1>Welcome to Student Management System</h1>  
    <a class="button" href="/auth/login">Login</a>  
  </div>  
</body>  
</html>
```



**`views/login.ejs`:**

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Login</title>  
  <link rel="stylesheet" href="/styles.css">  
</head>  
<body>  
  <div class="container">  
    <h1>Login</h1>  
    <form action="/auth/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>
    <% if (error) { %> <!-- Check if error is defined -->
        <p class="error"><%= error %></p>
    <% } %>
</div>
</body>
</html>

```

**`views/students.ejs`:**

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Student List</title>
    <link rel="stylesheet" href="/styles.css">
</head>
<body>
    <div class="container">
        <h1>Student List</h1>
        <a class="button" href="/logout">Logout</a>

        <h2>Add Student</h2>
        <form action="/students/add" method="POST">
            <input type="text" name="name" placeholder="Name" required>
            <input type="number" name="age" placeholder="Age" required>
            <input type="text" name="city" placeholder="City" required>
            <input type="text" name="className" placeholder="Class Name"
required>
            <input type="text" name="rollNo" placeholder="Roll No" required>

```

```

        <button type="submit">Add Student</button>
    </form>

    <h2>All Students</h2>
    <table>
        <thead>
            <tr>
                <th>Name</th>
                <th>Age</th>
                <th>City</th>
                <th>Class Name</th>
                <th>Roll No</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody>
            <% students.forEach(student => { %>
                <tr>
                    <td><%= student.name %></td>
                    <td><%= student.age %></td>
                    <td><%= student.city %></td>
                    <td><%= student.className %></td>
                    <td><%= student.rollNo %></td>
                    <td>
                        <a href="/students/edit/<%= student._id %>">Edit</a>
                        <a href="/students/delete/<%= student._id %>">Delete</a>
                    </td>
                </tr>
            <%
        </tbody>
    </table>
</div>
</body>

```

```
</html>
```

**`views/edit.ejs`:**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Edit Student</title>
  <link rel="stylesheet" href="/styles.css">
</head>
<body>
  <div class="container">
    <h1>Edit Student</h1>
    <form action="/students/edit/<%= student._id %>" method="POST">
      <input type="text" name="name" value="<%= student.name %>"
required>
      <input type="number" name="age" value="<%= student.age %>"
required>
      <input type="text" name="city" value="<%= student.city %>"
required>
      <input type="text" name="className" value="<%=
student.className %>" required>
      <input type="text" name="rollNo" value="<%= student.rollNo %>"
required>
      <button type="submit">Update Student</button>
    </form>
  </div>
</body>
</html>
```

**Step 8: Create CSS Styles**

**`public/styles.css`:**

```
body {
  font-family: Arial, sans-serif;
```

```
background-color: #f4f4f4;
margin: 0;
padding: 20px;
}

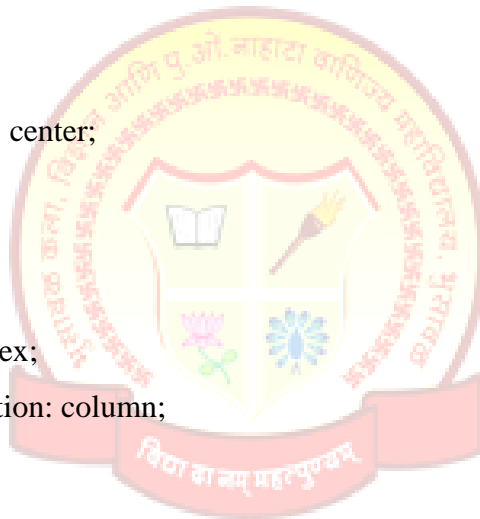
.container {
max-width: 800px;
margin: auto;
background: #fff;
padding: 20px;
border-radius: 5px;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
```

```
h1 {
text-align: center;
}
```

```
form {
display: flex;
flex-direction: column;
}
```

```
input[type="text"],
input[type="number"],
input[type="email"] {
padding: 10px;
margin: 10px 0;
border: 1px solid #ccc;
border-radius: 5px;
}
```

```
button {
padding: 10px;
background-color: #28a745;
```



```

    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

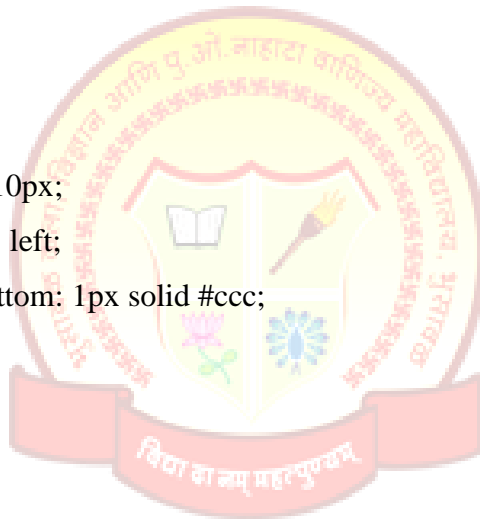
button:hover {
    background-color: #218838;
}

.table {
    width: 100%;
    border-collapse: collapse;
}

th, td {
    padding: 10px;
    text-align: left;
    border-bottom: 1px solid #ccc;
}

.error {
    color: red;
    text-align: center;
}

```



## Step 9: Run the Application

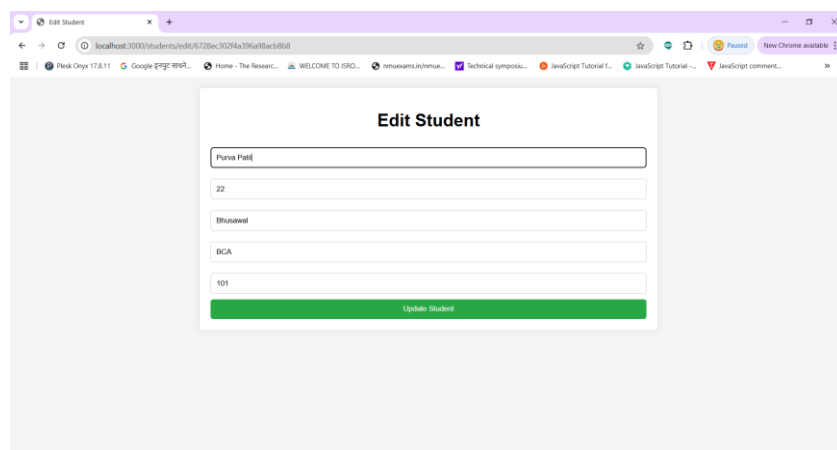
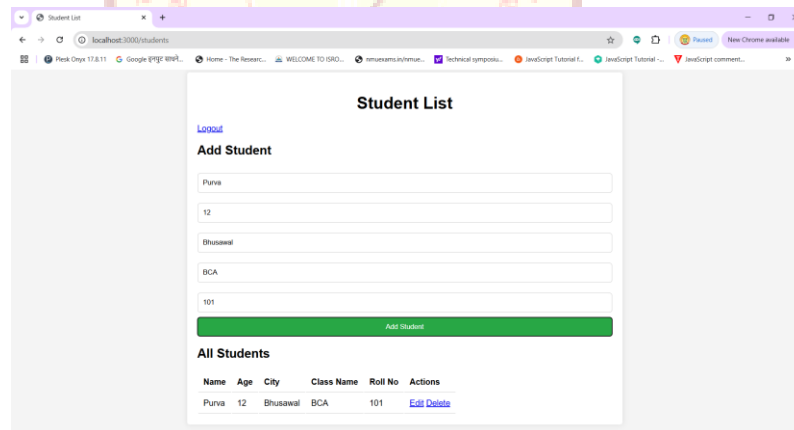
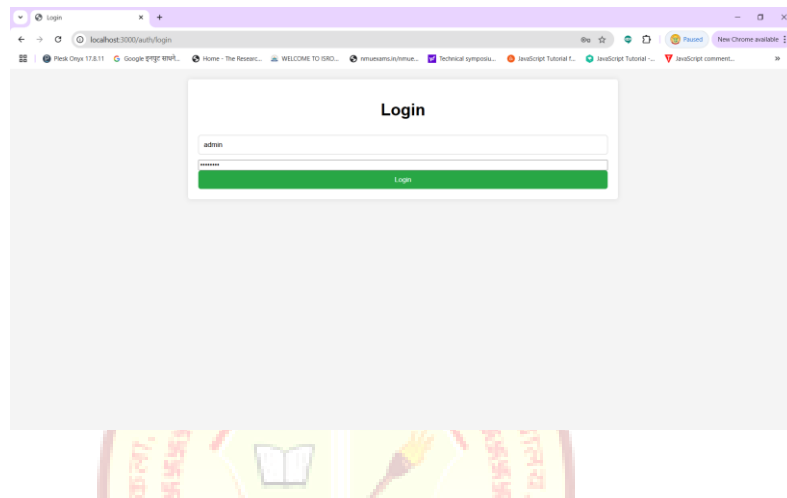
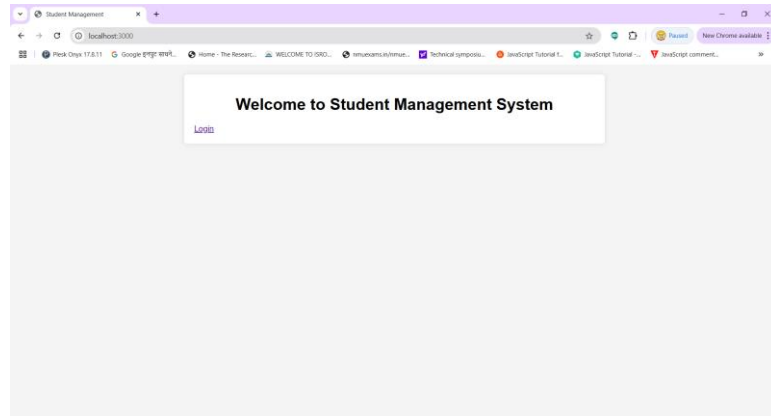
### 1. Start MongoDB (if not already running):

```
mongod
```

### 2. Run your application:

```
node app.js
```

### 3. Access the application at [http://localhost:3000]



Student List

[Logout](#)

Add Student

Add Student

All Students

Name	Age	City	Class Name	Roll No	Actions
Purva Harshal Patil	22	Bhusawal	BCA	101	<a href="#">Edit</a> <a href="#">Delete</a>

Student List

[Logout](#)

Add Student

Add Student

All Students

Name	Age	City	Class Name	Roll No	Actions
Pallavi Harshal Patil	35	Pune	MCA	256	<a href="#">Edit</a> <a href="#">Delete</a>





## Assignments – 13 Create Node JS Application to display student information.

### Step 1: Set Up the Node.js Server

1. Initialize the Node.js Project:

```
mkdir student-info-server
```

```
cd student-info-server
```

```
npm init -y
```

2. Install Dependencies:

```
npm install express mongoose cors
```

3. Create the Server:

Create a file named `server.js`:

```
// server.js
```

```
const express = require('express');
```

```
const mongoose = require('mongoose');
```

```
const cors = require('cors');
```

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

```
// MongoDB connection
```

```
mongoose.connect('mongodb://localhost:27017/studentdb', {  
  useNewUrlParser: true,  
  useUnifiedTopology: true,  
});
```

```
const db = mongoose.connection;
```

```
db.on('error', console.error.bind(console, 'connection error:'));
```

```
db.once('open', () => {
```

```
  console.log('Connected to MongoDB');
```

```
});
```

```
// Middleware
```

```
app.use(cors());
```

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

```
// Student Schema
```

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

```

    name: String,
    age: Number,
    city: String,
    className: String,
    rollNo: Number,
  });

const Student = mongoose.model('Student', studentSchema);

// Routes
app.get('/students', async (req, res) => {
  try {
    const students = await Student.find();
    res.json(students);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});

// Start server
app.listen(3001, () => {
  console.log('Server is running on port 3001');
});

```

## Step 2: Set Up the React Application

1. Initialize the React Project:

```

npx create-react-app student-info-client
cd student-info-client

```

2. Install Axios:

```

npm install axios

```

3. Create the Frontend Component:

Open `src/App.js` and replace its content with the following code:

```

// src/App.js
import React, { useState, useEffect } from 'react';
import axios from 'axios';

```

```

import './App.css';

function App() {
  const [students, setStudents] = useState([]);

  useEffect(() => {
    axios.get('http://localhost:3001/students')
      .then(response => {
        setStudents(response.data);
      })
      .catch(error => {
        console.error("There was an error fetching the students!",
error);
      });
  }, []);

  return (
    <div className="App">
      <h1>Student Information</h1>
      <ul>
        {students.map(student => (
          <li key={ student._id}>
            {student.name} - {student.age} - {student.city} -
{student.className} - {student.rollNo}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default App;

```

#### 4. Style the Application:

Create a `src/App.css` file with basic styles:

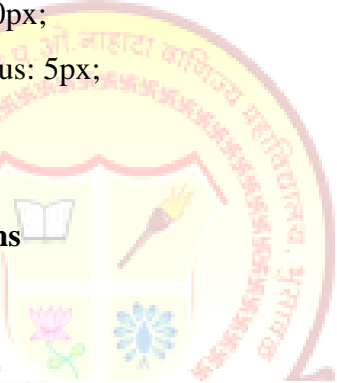
padding: 10px;  
border-radius: 5px;

**Applications**

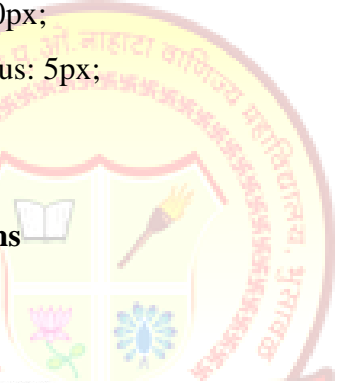
Server:

le server.js

- 

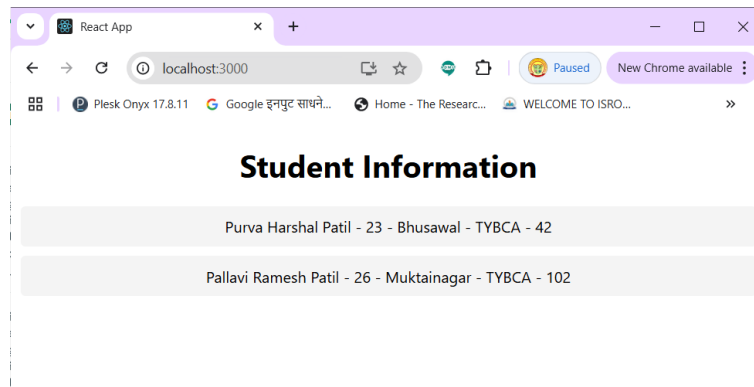
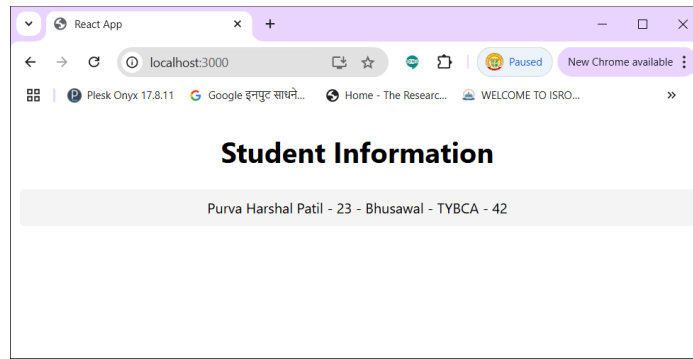


- 



-

Open your browser and navigate to `http://localhost:3000`.



## Assignments – 14 Create Node JS Application to insert, update, display and delete student information.

To create a Node.js GUI application for managing student information with MongoDB, we will use a combination of technologies:

- Node.js for the backend server.
- Express.js for the web framework.
- MongoDB for the database.
- Mongoose as the ODM (Object Data Modeling) library for MongoDB.
- HTML/CSS/JavaScript for the frontend.
- EJS (Embedded JavaScript) for templating.

### Step - 1 Set Up the Project

1. Initialize the project:

```
mkdir student-management-hvp  
cd student-management-hvp  
npm init -y
```

2. Install dependencies:

```
npm install express mongoose body-parser ejs
```

### Step - 2 Set Up MongoDB Connection with Mongoose

1. Create a file named db.js to handle the MongoDB connection:

```
// db.js  
// db.js  
  
const mongoose = require('mongoose');  
mongoose.connect('mongodb://localhost:27017/studentdb', {  
  useNewUrlParser: true,  
  useUnifiedTopology: true  
});  
  
const studentSchema = new mongoose.Schema({  
  name: String,  
  age: Number,  
  city: String,  
  className: String,  
  rollNo: Number  
});
```

```
const Student = mongoose.model('Student', studentSchema);
module.exports = Student;
```

### Step 3: Create the Express Server

1. Create a file named server.js for the Express server::

```
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const Student = require('./db');
const app = express();

// Middleware to parse request body
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json()); // Add this line to support JSON-encoded
bodies
app.set('view engine', 'ejs');
app.use(express.static('public'));
// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/studentdb', {
  useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => console.log('Connected to MongoDB'))
  .catch(err => console.log(err));
// Routes
app.get('/', async (req, res) => {
  try {
    const students = await Student.find({ });
    res.render('index', { students: students });
  } catch (err) {
    console.log(err);
    res.status(500).send('Internal Server Error');
  }
});
app.post('/addStudent', async (req, res) => {
  try {
    console.log(req.body); // Log request body to debug
    const newStudent = new Student({
```

```

    name: req.body.name,
    age: req.body.age,
    city: req.body.city,
    className: req.body.className,
    rollNo: req.body.rollNo
  });
  await newStudent.save();
  res.redirect('/');
} catch (err) {
  console.log(err);
  res.status(500).send('Internal Server Error');
}
});

app.post('/updateStudent', async (req, res) => {
  const studentId = req.body.id;
  try {
    await Student.findByIdAndUpdate(studentId, {
      name: req.body.name,
      age: req.body.age,
      city: req.body.city,
      className: req.body.className,
      rollNo: req.body.rollNo
    });
    res.redirect('/');
  } catch (err) {
    console.log(err);
    res.status(500).send('Internal Server Error');
  }
});

app.post('/deleteStudent', async (req, res) => {
  const studentId = req.body.id;
  try {
    console.log(`Attempting to delete student with ID: ${studentId}`); //
    Log student ID to debug
    await Student.findByIdAndDelete(studentId);

```



```

    res.redirect('/');
  } catch (err) {
    console.log(err);
    res.status(500).send('Internal Server Error');
  }
});
app.listen(3000, () => {
  console.log('Server started on port 3000');
});

```

#### Step 4: Create the Frontend

1. Create a directory named views and add an index.ejs file:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Student Management</title>
  <link rel="stylesheet" href="/css/styles.css">
</head>
<body>
  <h1>Student Management System</h1>

  <!-- Add Student Form -->
  <form id="addStudentForm" action="/addStudent" method="POST">
    <input type="text" name="name" placeholder="Name" required>
    <input type="number" name="age" placeholder="Age" required>
    <input type="text" name="city" placeholder="City" required>
    <input type="text" name="className" placeholder="Class"
required>
    <input type="number" name="rollNo" placeholder="Roll No"
required>
    <button type="submit">Add Student</button>
  </form>

```

```

<hr>
<!-- Display Students -->
<h2>Student List</h2>
<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>City</th>
<th>Class</th>
<th>Roll No</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<% students.forEach(student => { %>
<tr>
<td><%= student.name %></td>
<td><%= student.age %></td>
<td><%= student.city %></td>
<td><%= student.className %></td>
<td><%= student.rollNo %></td>
<td>
<form class="updateStudentForm" action="/updateStudent"
method="POST">
<input type="hidden" name="id" value="<%= student._id
%>">
<input type="text" name="name" value="<%= student.name
%>" required>
<input type="number" name="age" value="<%= student.age
%>" required>
<input type="text" name="city" value="<%= student.city %>"
required>
<input type="text" name="className" value="<%=
student.className %>" required>

```

```

        <input type="number" name="rollNo" value="<%=
student.rollNo %>" required>
        <button type="submit">Update</button>
    </form>
    <form class="deleteStudentForm" action="/deleteStudent"
method="POST">
        <input type="hidden" name="id" value="<%= student._id
%>">
        <button type="submit">Delete</button>
    </form>
</td>
</tr>
<% }); %>
</tbody>
</table>
<script>
    // Add event listeners to all insert forms

document.getElementById('addStudentForm').addEventListener('subm
it', function(event) {
    event.preventDefault();
    const form = this;

    fetch(form.action, {
        method: form.method,
        body: new URLSearchParams(new FormData(form))
    }).then(response => {
        if (response.ok) {
            alert('Student successfully inserted!');
            form.reset();
            window.location.reload(); // Reload the page to see the new
student in the list
        } else {
            alert('Failed to insert student. Please try again.');
```

```

    }).catch(error => {
      console.error('Error:', error);
      alert('Failed to insert student. Please try again.');
```

```

    });

    // Add event listeners to all update forms

```

```

    document.querySelectorAll('.updateStudentForm').forEach(form =>
    {
      form.addEventListener('submit', function(event) {
        event.preventDefault();
        const form = this;

        fetch(form.action, {
          method: form.method,
          body: new URLSearchParams(new FormData(form))
        }).then(response => {
          if (response.ok) {
            alert('Student successfully Updated!');
            window.location.reload(); // Reload the page to see the updated
list
          } else {
            alert('Failed to Updated student. Please try again.');
```

```

          }
        }).catch(error => {
          console.error('Error:', error);
          alert('Failed to Updated student. Please try again.');
```

```

        });
      });
    });

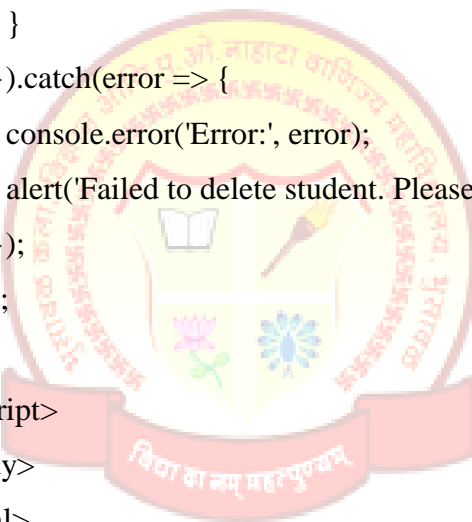
    // Add event listeners to all delete forms
    document.querySelectorAll('.deleteStudentForm').forEach(form =>
    {
      form.addEventListener('submit', function(event) {

```

```

event.preventDefault();
const form = this;

fetch(form.action, {
  method: form.method,
  body: new URLSearchParams(new FormData(form))
}).then(response => {
  if (response.ok) {
    alert('Student successfully deleted!');
    window.location.reload(); // Reload the page to see the updated
list
  } else {
    alert('Failed to delete student. Please try again.');
```



2. Create a directory named public/css and add a styles.css file:

```

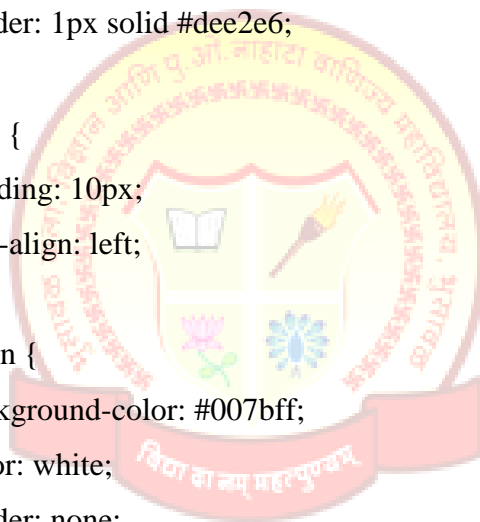
/* public/css/styles.css */
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 20px;
  background-color: #f8f9fa;
}
h1, h2 {
  color: #343a40;
}
form {
```

```

margin-bottom: 20px;
}
input, button {
padding: 5px;
margin: 5px;
font-size: 12px;
}
table {
width: 100%;
border-collapse: collapse;
}

table, th, td {
border: 1px solid #dee2e6;
}
th, td {
padding: 10px;
text-align: left;
}
button {
background-color: #007bff;
color: white;
border: none;
cursor: pointer;
}
button:hover {
background-color: #0056b3;
}

```

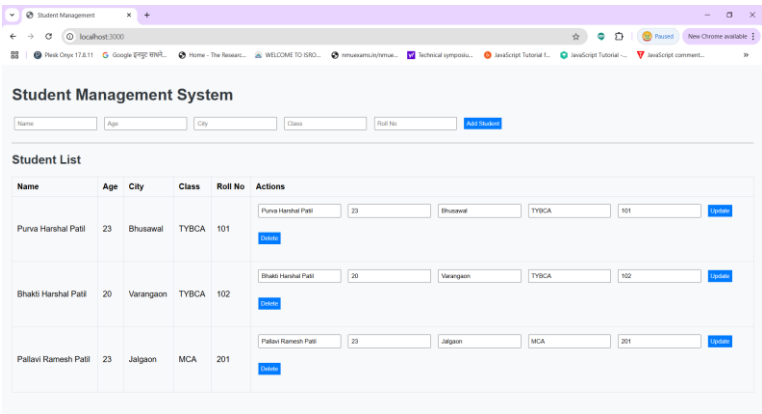
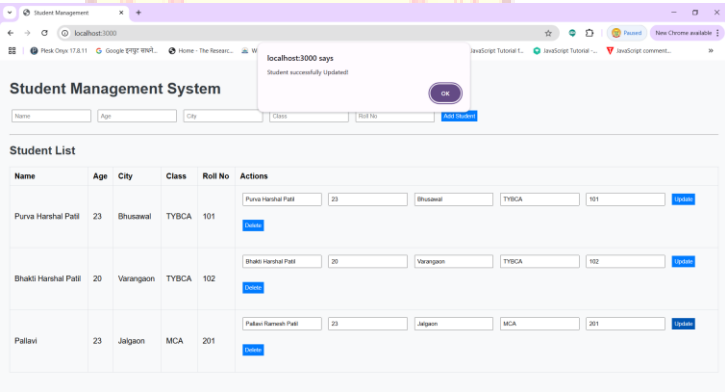
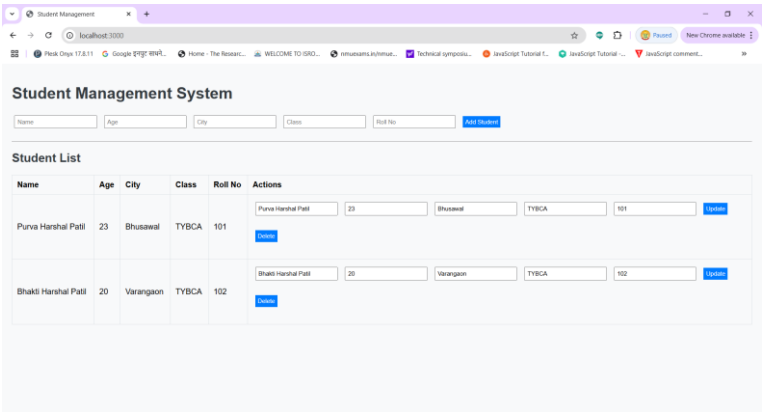
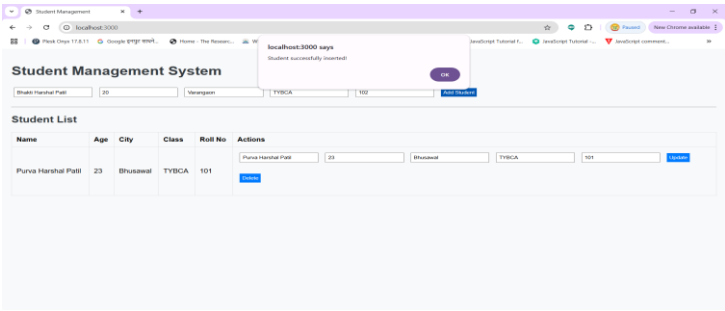


## Step - 5 Running the Application

1. Run your Node.js application:

**node server.js**

2. Open your browser and navigate to `http://localhost:3000`.



Student Management

localhost:3000

localhost3000 says  
Student successfully deleted!

OK

Name

Age

City

Class

Roll No

Add Student

Student List

Name	Age	City	Class	Roll No	Actions
Purva Harshal Patil	23	Bhusawal	TYBCA	101	<div><div>Purva Harshal Patil</div><div>23</div><div>Bhusawal</div><div>TYBCA</div><div>101</div><div>Update</div></div> <div>Delete</div>
Bhakti Harshal Patil	20	Varangaon	TYBCA	102	<div><div>Bhakti Harshal Patil</div><div>20</div><div>Varangaon</div><div>TYBCA</div><div>102</div><div>Update</div></div> <div>Delete</div>
Pallavi Ramesh Patil	23	Jalgaon	MCA	201	<div><div>Pallavi Ramesh Patil</div><div>23</div><div>Jalgaon</div><div>MCA</div><div>201</div><div>Update</div></div> <div>Delete</div>

Student Management

localhost:3000

Student Management System

Name

Age

City

Class

Roll No

Add Student

Student List

Name	Age	City	Class	Roll No	Actions
Purva Harshal Patil	23	Bhusawal	TYBCA	101	<div><div>Purva Harshal Patil</div><div>23</div><div>Bhusawal</div><div>TYBCA</div><div>101</div><div>Update</div></div> <div>Delete</div>
Bhakti Harshal Patil	20	Varangaon	TYBCA	102	<div><div>Bhakti Harshal Patil</div><div>20</div><div>Varangaon</div><div>TYBCA</div><div>102</div><div>Update</div></div> <div>Delete</div>

