

Title : 1 Setting Up React Environment and Creating a Basic React Application
. (Install Node.js, NPM, Create React App)

Title : 2. JSX and Component Creation Using Props.

App.jsx :

```
import Greeting from './Greeting'

const App = () => {
  return (
    <div>
      <Greeting name="Aaditya" />
      <Greeting name="Pranav" />
      <Greeting name="Parth" />
    </div>
  )
}

export default App
```

Greeting.jsx :

```
const Greeting = (props) => {
  return (
    <div>
      <h3>Hello, {props.name}...!!!</h3>
    </div>
  )
}

export default Greeting
```

Output :

Title : 3. State Management in React Using use State: Building a Counter App

App.jsx :

```
import { useState } from 'react'

const App = () => {
  const [count, setCount] = useState(0)
  return (
    <div className="flex flex-col justify-center items-center m-2 w-3/5 bg-amber-100 rounded-2xl">
      <h1 className='m-2 text-5xl font-bold'>React Counter</h1>
      <div className='p-5 border-2 rounded w-10 m-2 h-8 flex justify-center items-center mt-4'>{count}</div>
    </div>
  )
}

export default App
```

```

        <div className='flex gap-10 m-4'>
          <button className='border-2 rounded px-5'
            onClick={()=> setCount(count+1)}>+</button>
          <button className='border-2 rounded px-5'
            onClick={()=> setCount(count-1)}>-</button>
          <button className='border-2 rounded px-5'
            onClick={()=> setCount(0)}>Reset</button>
        </div>
      </div>
    )
}

export default App

```

Output :

Title : 4 Event Handling in React: Managing Button Clicks and Input Changes

App.jsx :

```

import { useState } from 'react'

const App = () => {
  const [name, setname] = useState('')
  const [message, setmessage] = useState('')
  const changeName= (e)=>{
    console.log(e.target.value);

    setname(e.target.value)
  }
  const buttonClick = ()=>{
    setmessage(`Hello, ${name}`)
  }
  return (
    <div className='m-10'>
      <input
        type="text"
        placeholder='Enter Your Name...'
        value={name}
        onChange={changeName}
        className='border-2 rounded m-4'/> <br />

      <button onClick={buttonClick}
        className='border-2 rounded m-4 px-2'>Greet Me</button>
      <br />
      <p className='m-4 text-emerald-500 text-xl font-
bold'>{message}</p>
    </div>
  )
}

export default App

```

Output :

Title : 5. Rendering Dynamic Lists in React Using Array Mapping and Key Props

App.jsx :

```
import React from 'react'

const App = () => {
  const fruits = ['Apple', 'Banana', 'Cherry'];// array
  return (
    <div className='m-4'>
      <h1>FRUITS LIST</h1>
      <ul>
        {fruits.map((fruit, index) => (
          <li key={index}> {fruit}</li>
        ))}
      </ul>
    </div>
  )
}

export default App
```

Output :

Title : 6 Building Controlled Forms in React: Managing State and Implementing Basic Validation

App.jsx :

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

const App = () => {
  const [inputValue, setInputValue] = useState('');
  const [submit, setSubmit] = useState("");
  const [error, setError] = useState("");

  function handleInputChange(event) {
    const value = event.target.value;
    setInputValue(value);
    if (value.trim() !== "") {
      setError("");
    }
  }

  function handleSubmit(event) {
    event.preventDefault();
    if (inputValue.trim() === "") {
      setError("Name is required.");
      setSubmit("");
      return;
    }
    setSubmit(`Hello , ${inputValue} your form is submitted`);
    setError("");
  }

  return (
    <form onSubmit={handleSubmit} style={{ margin: "50px" }}>
      <label>
        Name:
        <input
          type="text"
          value={inputValue}
          onChange={handleInputChange}
        />
      </label>
      <br />
      <button type="submit" className="border rounded bg-fuchsia-300 px-4 my-4">
        Submit
      </button>
      {error && <p style={{ color: 'red' }}>{error}</p>}
      <p>{submit}</p>
    </form>
  );
}

export default App
```

Output :

Title : 7 Styling React Components: Implementing CSS Modules and Inline Styles

App.jsx :

```
import styles from './Button.module.css';

const App = () => {
  return (
    <button
      className={styles.button} //module css
      style={{ backgroundColor: "rgba(154, 43, 252, 0.655)" }} //inline
      css
      onClick={() => { alert("Button Clicked") }}>
      Click Me
    </button>
  );
}
export default App
```

Button.module.css :

```
.button {
  color: white;
  padding: 10px 20px;
  margin: 10px;
  font-weight: 500
}
```

Output :

Title : 8 Conditional styling in React using both CSS Modules and inline styles.
(Toggle Button with Conditional Styling).

App.jsx :

```
import './toggle.css';
import { useState } from "react";
const App = () => {
  const [isOn, setIsOn] = useState(false);

  const buttonStyle = {
    backgroundColor: isOn ? '#4CAF50' : '#f44336',
    color: 'white',
    padding: '12px 20px',
    border: 'none',
    borderRadius: '5px',
    fontSize: '16px',
    cursor: 'pointer',
    transition: 'background-color 0.3s ease',
  };

  return (
    <div className="container">
      <span className="label">Toggle is {isOn ? 'ON' : 'OFF'}</span>
      <button style={buttonStyle} onClick={() => setIsOn(!isOn)}>
        {isOn ? 'Turn OFF' : 'Turn ON'}
      </button>
    </div>
  );
}

export default App
```

Output :

Title : 9 Developing a Personal Portfolio Website with React and Bootstrap

App.jsx :

```
import Navbar from './Navbar';
import Home from './Home';
import About from './About';
import Projects from './Projects';
import Footer from './Footer';
function App() {
  return (
    <div>
      <Navbar />
      <Home />
      <About />
      <Projects />
```

```
        <Footer />
    </div>
);
}
export default App;
```

Navbar.jsx:

```
function Navbar() {
    return (
        <nav className="navbar navbar-expand-lg navbar-dark bg-dark">
            <div className="container">
                <a className="navbar-brand">My Portfolio</a>
                <div className="collapse navbar-collapse">
                    <ul className="navbar-nav ms-auto">
                        <li className="nav-item"><a className="nav-link" href="#home">Home</a></li>
                        <li className="nav-item"><a className="nav-link" href="#about">About</a></li>
                        <li className="nav-item"><a className="nav-link" href="#projects">Projects</a></li>
                    </ul>
                </div>
            </div>
        </nav>
    );
}
export default Navbar;
```

Home.jsx

```
function Home() {
  return (
    <section id="home" className="bg-light text-center py-5">
      <div className="container">
        <h1>Hello, I'm Aaditya </h1>
        <p>Frontend Developer | React Enthusiast</p>
      </div>
    </section>
  );
}
export default Home;
```

About.jsx

```
import React from 'react';
function About() {
  return (
    <section id="about" className="py-5">
      <div className="container">
        <h2>About Me</h2>
        <p>I am a passionate web developer with skills in
React, Bootstrap, and modern web
technologies.</p>
      </div>
    </section>
  );
}
export default About;
```

Project.jsx

```
function Projects() {
  return (
    <section id="projects" className="bg-light py-5">
      <div className="container">
        <h2>Projects</h2>
        <ul>
          <li>Portfolio Website</li>
          <li>To-do App with React</li>
          <li>Weather App using API</li>
        </ul>
      </div>
    </section>
  );
}
export default Projects;
```

Footer.jsx

```
function Footer() {  
  return (  
    <footer className="bg-dark text-white text-center py-3">  
      <p>&copy; 2025 My Portfolio. All rights reserved.</p>  
    </footer>  
  );  
}  
export default Footer;
```

Output :

Title : 10 Setting Up a Node.js and Express.js Development Environment: A Practical Guide

Step 1: Setup the Backend (Node.js + Express.js)

1. Go to your folder on the top write cmd
2. In cmd write this commands one by one

```
mkdir fullstack-app
cd fullstack-app
mkdir backend
cd backend
npm init -y
code .
in VS code your folder is open and create file server.js copy the code in server.js
```

2. Install Express

```
npm install express cors
```

3. Create server.js

add this code in server.js

```
const express = require('express');
const cors = require('cors');
const app = express();
const PORT = 5000;
app.use(cors());
app.use(express.json());
// Dummy API
app.get('/api/message', (req, res) => {
  res.json({ message: "Hello from the backend!" });
});
app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

4. Run the Backend Server

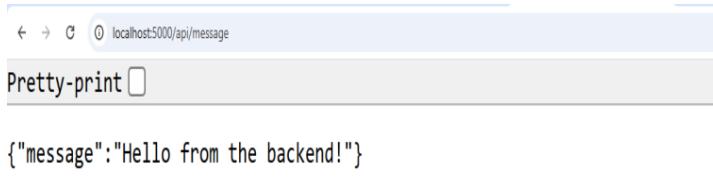
Run this command on CMD :

```
node server.js
```

Visit: <http://localhost:5000/api/message>

You should see:

```
{ "message": "Hello from the backend!" }
```



A screenshot of a browser window. The address bar shows 'localhost:5000/api/message'. Below the address bar is a 'Pretty-print' button. The main content area displays the JSON response: {"message": "Hello from the backend!"}

Practical no.11: Implementing Basic Routing in Express.js: Creating Multiple GET Endpoints.

Step: 1:- Create folder on desktop.Then open folder and open cmd in that folder.

Step: 2:- npm init -y

Step: 3:- npm install express then code .

Step: 4:- Create Index.js in vs code and type this code.

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

const port = 3000;

app.get('/', (req, res) => {
  res.send('Welcome to the My Program!');
});

app.get('/about', (req, res) => {
  res.send('This is the About Web Page.');
});

app.get('/contact', (req, res) => {
  res.send('Contact us at aiabi45@gmail.com');
});

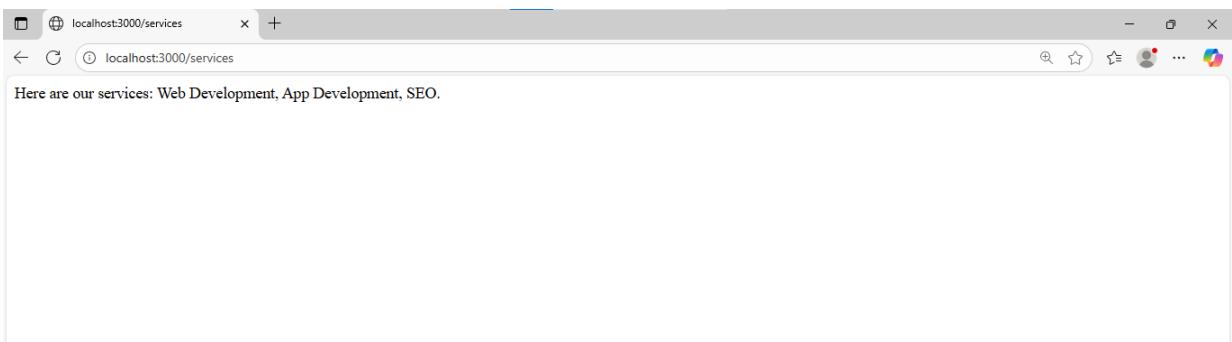
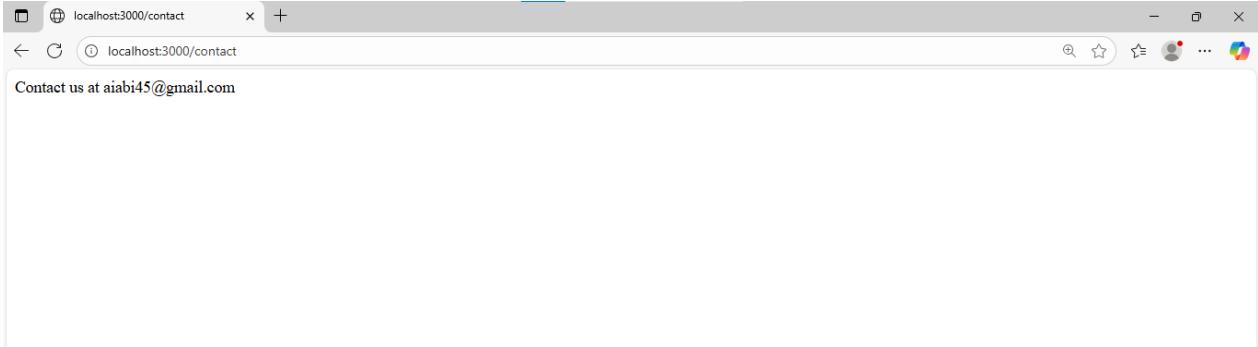
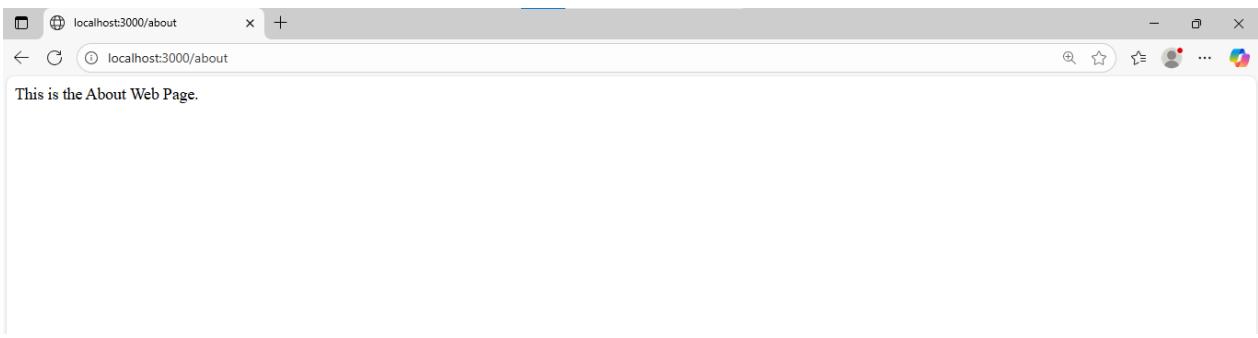
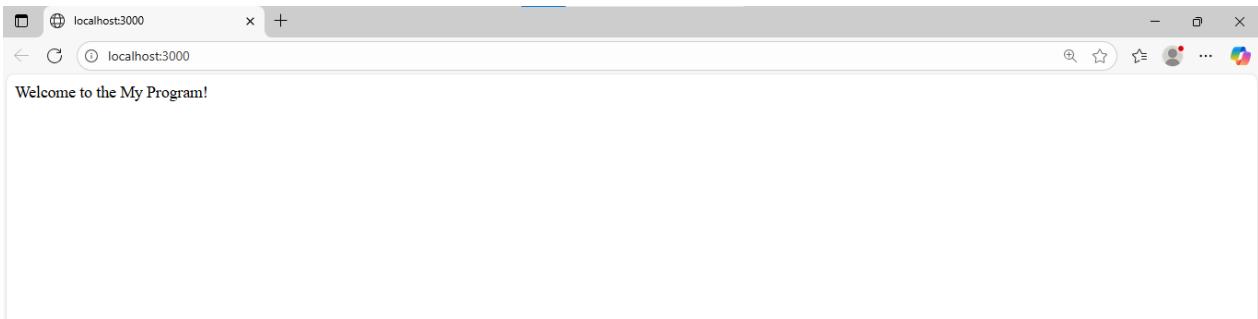
app.get('/services', (req, res) => {
  res.send('Here are our services: Web Development, App Development, SEO.');
});

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

Step 5: Run this in terminal

```
node index.js
```

Output:



Name: Aditya Dnyaneshwar Pawar Roll no: 178 Class: SYBCA
Practical no.12- Building a RESTful API with Express.js: A Hands-On Guide to HTTP Methods on (Customer details)

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

let customers = [
  { id: 1, name: 'OM', email: 'Om@example.com' },
  { id: 2, name: 'Lalit', email: 'lalit424@example.com' }
];

app.get('/customers', (req, res) => {
  res.json(customers);
});

app.get('/customers/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const customer = customers.find(c => c.id === id);
  if (!customer) {
    return res.status(404).send('Customer not found');
  }
  res.json(customer);
});

app.post('/customers', (req, res) => {
  const { name, email } = req.body;
  const newCustomer = {
    id: customers.length + 1,
    name,
    email
  };
  customers.push(newCustomer);
  res.status(201).json(newCustomer);
});

app.put('/customers/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const customer = customers.find(c => c.id === id);
  if (!customer) {
    return res.status(404).send('Customer not found');
  }
})
```

```

customer.name = req.body.name || customer.name;
customer.email = req.body.email || customer.email;

res.json(customer);
});

app.delete('/customers/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const index = customers.findIndex(c => c.id === id);
  if (index === -1) {
    return res.status(404).send('Customer not found');
  }
  const deletedCustomer = customers.splice(index, 1);
  res.json(deletedCustomer);
});

app.listen(port, () => {
  console.log(`Customer API is running at http://localhost:${port}`);
});

```

1. GET all customers

URL: <http://localhost:3000/customers>

Method: GET

The screenshot shows a Postman interface with a request and response pane.

Request (Left Panel):

- Method: GET
- URL: <http://localhost:3000/customers>
- Headers (2): Content-Type, Accept
- Auth: None
- Body: None
- Tests: None
- Pre Run: None

Response (Right Panel):

- Status: 200 OK
- Size: 102 Bytes
- Time: 57 ms
- Headers (6): Content-Type, Content-Length, Date, ETag, Vary, X-Powered-By
- Cookies: None
- Results: JSON array containing two customer objects.
- Docs: None

```

1  [
2   {
3     "id": 1,
4     "name": "OM",
5     "email": "Om@example.com"
6   },
7   {
8     "id": 2,
9     "name": "Lalit",
10    "email": "lalit424@example.com"
11  }
12 ]

```

2. GET a customer by ID

URL: <http://localhost:3000/customers/1>

Method: GET

The screenshot shows the Postman interface with a successful HTTP request. The request details are as follows:

- Method:** GET
- URL:** <http://localhost:3000/customers/1>
- Headers:** 2 (Auth, Body)
- Tests:** None
- Pre Run:** None

The response details are as follows:

- Status:** 200 OK
- Size:** 45 Bytes
- Time:** 6 ms

The response body is a JSON object:

```
1 {  
2   "id": 1,  
3   "name": "OM",  
4   "email": "Om@example.com"  
5 }
```

3. POST a new customer

URL: <http://localhost:3000/customers>

Method: POST

The screenshot shows the Postman interface with a successful HTTP request. The request details are as follows:

- Method:** POST
- URL:** <http://localhost:3000/customers>
- Headers:** 2 (Auth, Body)
- Body:** 1 (JSON)
- Tests:** None
- Pre Run:** None

The response details are as follows:

- Status:** 201 Created
- Size:** 54 Bytes
- Time:** 63 ms

The response body is a JSON object:

```
1 {  
2   "id": 3,  
3   "name": "Abhay",  
4   "email": "abhay125@example.com"  
5 }
```

4. PUT (update) a customer

URL: <http://localhost:3000/customers/2>

Method: PUT

The screenshot shows the Postman interface with a dark theme. A new request is being prepared for the URL <http://localhost:3000/customers/1>. The method is set to PUT. The Body tab is selected, showing JSON content with the following payload:

```
1 {  
2   "name": "Rohit Updated",  
3   "email": "rohit@example.com"  
4 }
```

The response section shows a successful 200 OK status with a size of 59 bytes and a time of 4 ms. The response body is identical to the request payload.

5. DELETE a customer

URL: <http://localhost:3000/customers/1>

Method: DELETE

The screenshot shows the Postman interface with a dark theme. A new request is being prepared for the URL <http://localhost:3000/customers/1>. The method is set to DELETE. The Body tab is selected, showing an empty JSON content block.

The response section shows a successful 200 OK status with a size of 61 bytes and a time of 6 ms. The response body is a JSON array containing one element, which is identical to the request payload.

Name: Aditya Dnyaneshwar Pawar Roll no: 178 Class: SYBCA

Practical 13 :- Building RESTful APIs: Managing JSON POST Requests in Express.js (Employee Post Method).

1. Create a new folder for your project and open it in terminal.

2. Run: npm init -y

npm install express

Code:-

```
const express = require('express');
const app = express();
const port = 3000;
app.use(express.json());
let employees = [
  { id: 1, name: 'Alice', role: 'Developer' },
  { id: 2, name: 'Bob', role: 'Designer' }
];
app.get('/employees', (req, res) => {
  res.json(employees);
});
app.post('/employees', (req, res) => {
  const newEmployee = req.body;
  if (!newEmployee.name || !newEmployee.role) {
    return res.status(400).json({ error: 'Name and role are required' });
  }
  newEmployee.id = employees.length + 1;
  employees.push(newEmployee);
});
```

```

    res.status(201).json(newEmployee);
  });

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

```

Output:-

Run your server

Run in terminal: node index.js

You should see: Server running at <http://localhost:3000>

Open your browser or Postman and visit:

<http://localhost:3000/employees>

GET:

The screenshot shows the Thunder Client interface. In the top bar, there are tabs for 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', etc. Below the tabs, there's a search bar and a toolbar with icons for 'New Request', 'Activity', 'Collections', 'Env', and 'filter activity'. The main area has a sidebar on the left with a tree view showing a POST request to 'localhost:3000/employees' made 9 mins ago and a GET request to 'localhost:3000/employees' made 12 mins ago. The central workspace shows a 'New Request' dialog for a POST to 'http://localhost:3000/employees'. The 'Body' tab is selected, showing a JSON content pane with the following code:

```

1  {
2    "name": "Charlie",
3    "role": "Manager",
4    "id": 3
5  }

```

To the right of the workspace, there's a status bar showing 'Status: 201 Created', 'Size: 42 Bytes', and 'Time: 16 ms'. Below the status bar, there are tabs for 'Response', 'Headers', 'Cookies', 'Results', and 'Docs'. At the bottom of the screen, there's a taskbar with various icons and a system tray showing the date and time.

POST:-

Use Postman or curl to send a POST request to: <http://localhost:3000/employees>

With this JSON body:

```
{  
  "name": "Charlie",  
  "role": "Manager"  
}
```

The screenshot shows the Thunder Client application interface. On the left, there's a sidebar with icons for file operations, a search bar, and a list of recent requests. The main area has tabs for 'index.js' and 'New Request'. A 'New Request' tab is active, showing a POST request to 'http://localhost:3000/employees'. The 'Body' tab is selected, displaying JSON content:

```
POST /localhost:3000/employees  
{"name": "Charlie", "role": "Manager"}
```

The response tab shows a successful 200 OK status with a size of 84 bytes and a time of 12 ms. The response body is a JSON array:

```
[  
  {  
    "id": 1,  
    "name": "Alice",  
    "role": "Developer"  
  },  
  {  
    "id": 2,  
    "name": "Bob",  
    "role": "Designer"  
  }]
```

The bottom of the screen shows the Windows taskbar with various pinned icons and the system tray.

Practical no.14: Implementing CRUD Operations (Students)Using RE/STful API and HTTP Methods.

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

let students = [];
let nextId = 1;
// CREATE - Add new student
app.post('/students', (req, res) => {
  const student = { id: nextId++, ...req.body };
  students.push(student);
  res.status(201).json({ message: 'Student created', student });
});

app.get('/students', (req, res) => {
  res.json(students);
});

app.get('/students/:id', (req, res) => {
  const student = students.find(s => s.id === req.params.id);
  if (!student) {
    return res.status(404).json({ message: 'Student not found' });
  }
  res.json(student);
});
```

```
app.put('/students/:id', (req, res) => {
  const index = students.findIndex(s => s.id === req.params.id);
  if (index === -1) {
    return res.status(404).json({ message: 'Student not found' });
  }
  students[index] = { id: students[index].id, ...req.body };
  res.json({ message: 'Student updated', student: students[index] });
});

app.delete('/students/:id', (req, res) => {
  const index = students.findIndex(s => s.id === req.params.id);
  if (index === -1) {
    return res.status(404).json({ message: 'Student not found' });
  }
  const removedStudent = students.splice(index, 1);
  res.json({ message: 'Student deleted', student: removedStudent[0] });
});

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

Output:

The screenshot shows the Thunder Client interface. In the top navigation bar, the tab 'POST /students' is selected. The URL field contains 'http://localhost:3000/students'. The 'Body' tab is active, showing JSON content:

```
1 {
2   "name": "John Doe",
3   "age": 20,
4   "course": "Math"
5 }
```

Below the body, the status is shown as 'Status: 201 Created Size: 91 Bytes Time: 35 ms'. The response tab shows the JSON response:

```
1 {
2   "message": "Student created",
3   "student": {
4     "id": 1,
5     "name": "John Doe",
6     "age": 20,
7     "course": "Math"
8   }
9 }
```

The left sidebar shows a collection named 'student' with a single item: 'POST POST /students just now'.

The screenshot shows the Thunder Client interface. In the top navigation bar, the tab 'GET /students' is selected. The URL field contains 'http://localhost:3000/students'. The 'Body' tab is active, showing JSON content:

```
1
```

Below the body, the status is shown as 'Status: 200 OK Size: 53 Bytes Time: 6 ms'. The response tab shows the JSON response:

```
1 [
2   {
3     "id": 1,
4     "name": "John Doe",
5     "age": 20,
6     "course": "Math"
7   }
8 ]
```

The left sidebar shows a collection named 'student' with two items: 'POST POST /students just now' and 'GET GET /students just now'.

The screenshot shows the Thunder Client interface. On the left sidebar, under the 'student' collection, there is a recent activity entry for a 'GET /students/1' request made just now. The main panel displays a successful GET request to `http://localhost:3000/students/1`. The 'Body' tab is selected, showing the JSON response:

```
1 {  
2   "id": 1,  
3   "name": "Johnny Doe",  
4   "age": 21,  
5   "course": "Physics"  
6 }
```

The status bar at the bottom indicates: Status: 200 OK, Size: 56 Bytes, Time: 4 ms.

The screenshot shows the Thunder Client interface. On the left sidebar, under the 'student' collection, there is a recent activity entry for a 'PUT /students/1' request made just now. The main panel displays a successful PUT request to `http://localhost:3000/students/1`. The 'Body' tab is selected, showing the JSON payload sent in the request:

```
1 {  
2   "name": "Johnny Doe",  
3   "age": 21,  
4   "course": "Physics"  
5 }
```

The status bar at the bottom indicates: Status: 200 OK, Size: 96 Bytes, Time: 3 ms.

THUNDER CLIENT

New Request

Activity Collections Env

filter collections

student

- POST** POST /students just now
- GET** GET /students just now
- GET** GET /students/1 just now
- PUT** PUT /students/1 just now
- GET** DELETE /students/1 just now

Welcome JS index.js TC POST /students TC GET /students TC GET /students/1 TC DELETE /students/1

GET http://localhost:3000/students/1

Query Headers 2 Auth Body Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content

```
1
```

Status: 200 OK Size: 56 Bytes Time: 5 ms

Response Headers 6 Cookies Results Docs

```
1 {
2   "id": 1,
3   "name": "Johnny Doe",
4   "age": 21,
5   "course": "Physics"
6 }
```

Type here to search



31°C ENG IN 3:08 PM 7/15/2024

PRACTICAL 15: SETTING UP MONGO DB ENVIRONMENT: TEACHERS API WITH MONGODB BY USING MONGODB COMPASS.

Server.js:

```
const express =  
require('express'); const  
mongoose =  
require('mongoose');  
const app = express();  
app.use(express.json()); // middleware to parse JSON  
  
// Connect to MongoDB  
mongoose.connect("mongodb://127.0.0.1:27017/  
MongoDB", { useNewUrlParser: true,  
useUnifiedTopology: true,  
})  
.then(() => console.log("Connected to MongoDB"))  
.catch(err => console.error(" MongoDB connection error:", err));  
  
// Create Teacher Schema  
const teacherSchema = new  
mongoose.Schema({ name: String,  
subject:  
String,  
email:  
String,  
experience:  
Number,  
available  
:  
Boolean  
});  
  
// Create Teacher Model  
const Teacher = mongoose.model('Teacher', teacherSchema);  
// Get all teachers  
app.get('/teachers', async (req,  
res) => { const teachers =  
await Teacher.find();  
res.json(teachers);  
});  
  
// Add new teacher  
app.post('/teachers', async  
(req, res) => { const teacher =  
new Teacher(req.body); await  
teacher.save();
```

```

        res.status(201).json(teacher);
    });

    // Get teacher by ID
    app.get('/teachers/:id', async (req, res) => {
        const teacher = await Teacher.findById(req.params.id);
        if (!teacher) return res.status(404).send('Teacher not found');
        res.json(teacher);
    });

    // Update teacher by ID
    app.put('/teachers/:id', async (req, res) => {
        const teacher = await Teacher.findByIdAndUpdate(req.params.id, req.body, { new: true });
        if (!teacher) return res.status(404).send('Teacher not found');
        res.json(teacher);
    });

    // Delete teacher by ID
    app.delete('/teachers/:id', async (req, res) => {
        const result = await Teacher.findByIdAndDelete(req.params.id);
        if (!result) return res.status(404).send('Teacher not found');
        res.send('Teacher deleted');
    });

    // Start the server
    const PORT = 3001;
    app.listen(PORT, () => {
        console.log(`Server is running at http://localhost:${PORT}`);
    });
}

```

Output:

POST

HTTP <http://localhost:3001/teachers>

POST <http://localhost:3001/teachers>

Params Authorization Headers (9) **Body** Scripts Settings

Body Cookies Headers (7) Test Results

Pretty Raw Preview JSON ↴

```
1 {
2   "name": "Poonam Mam",
3   "subject": "Web Dev III",
4   "email": "webdev@imr.com",
5   "experience": 5,
6   "available": true,
7   "_id": "68bfb765f98bfff6f018611cc",
8   "__v": 0
9 }
```

PUT

HTTP <http://localhost:3001/teachers/68bfb765f98bfff6f018611cc>

PUT <http://localhost:3001/teachers/68bfb765f98bfff6f018611cc>

Params Authorization Headers (9) **Body** Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON ↴

```
1 {
2   ... "name": "Rohini Mam"
3 }
```

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 15 ms

Pretty Raw Preview JSON ↴

```
1 {"_id": "68bfb765f98bfff6f018611cc", "name": "Rohini Mam", "subject": "Web Dev III", "email": "webdev@imr.com", "experience": 5, "available": true, "__v": 0}
```

DELETE

HTTP <http://localhost:3001/teachers/68bfb765f98bfff6f018611cc>

DELETE <http://localhost:3001/teachers/68bfb765f98bfff6f018611cc>

Params Authorization Headers (7) Body Scripts Settings

Body Cookies Headers (7) Test Results

Pretty Raw Preview HTML ↴

```
1 Teacher deleted
```


Practical no.16: Setting up Mongo DB Environment: Teachers API with MongoDB by using MongoDB

Step 1: Backend Setup

open folder and write this commands in cmd

```
mkdir inventory-mini-api  
cd inventory-mini-api  
npm init -y  
npm install express mongoose cors
```

Create server.js

```
const express = require('express');  
const mongoose = require('mongoose');  
const cors = require('cors');  
const app = express();  
app.use(cors());  
app.use(express.json());  
mongoose.connect('mongodb://127.0.0.1:27017/inventoryDB', {  
  useNewUrlParser: true,  
  useUnifiedTopology: true  
});  
mongoose.connection.once("open", () => {  
  console.log("⚡ Connected to MongoDB");  
});  
mongoose.connection.on("error", (err) => {  
  console.error("✗ MongoDB connection error:", err);  
});
```

```
const Item = mongoose.model('Item', new mongoose.Schema({
  name: String,
  quantity: Number
}));

app.get('/items', async (req, res) => {
  try {
    const items = await Item.find();
    res.json(items);
  } catch (err) {
    console.error("Error fetching items:", err);
    res.status(500).json({ error: "Failed to fetch items" });
  }
});

app.post('/items', async (req, res) => {
  try {
    const { name, quantity } = req.body;
    if (!name || !quantity) {
      return res.status(400).json({ error: "Name and Quantity required" });
    }
    const item = new Item({ name, quantity });
    await item.save();
    res.json(item);
  } catch (err) {
    console.error("Error saving item:", err);
    res.status(500).json({ error: "Failed to save item" });
  }
});

app.delete('/items/:id', async (req, res) => {
```

```

try {

  await Item.findByIdAndDelete(req.params.id);

  res.json({ message: "Item deleted" });

} catch (err) {

  console.error("Error deleting item:", err);

  res.status(500).json({ error: "Failed to delete item" });

}

});

app.listen(5000, () => console.log('⚡ Backend running on http://localhost:5000'));

```

⌚ Frontend: React

Step 2: Frontend Setup

Open new terminal in VS Code and run this commands

Create React App

npx create-react-app inventory-mini-client

cd inventory-mini-client

npm install axios

/App.js will be created :

// src/App.js

```

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

import axios from 'axios';

const API = "http://localhost:5000/items";

function App() {

  const [items, setItems] = useState([]);

  const [name, setName] = useState("");

  const [quantity, setQuantity] = useState(1);

  const loadItems = async () => setItems((await axios.get(API)).data);

  const addItem = async () => {

    await axios.post(API, { name, quantity });
  }
}

```

```
setName("");
setQuantity(1);
loadItems();
};

const deleteItem = async (id) => {
await axios.delete(`$API}/${id}`);
loadItems();
};

useEffect(() => { loadItems(); }, []);
return (
<div style={{ padding: 20 }}>
<h2>📝 Inventory</h2>
<input value={name} onChange={e => setName(e.target.value)} placeholder="Item Name" />
<input type="number" value={quantity} onChange={e => setQuantity(Number(e.target.value))}>
<button onClick={addItem}>Add</button>
<ul>
{items.map(item => (
<li key={item._id}>
{item.name} - Qty: {item.quantity}
<button onClick={() => deleteItem(item._id)}>✖</button>
</li>
))}
</ul>
</div>
);
}

export default App;
```

□ Run the App

Backend:

```
cd inventory-mini-api
```

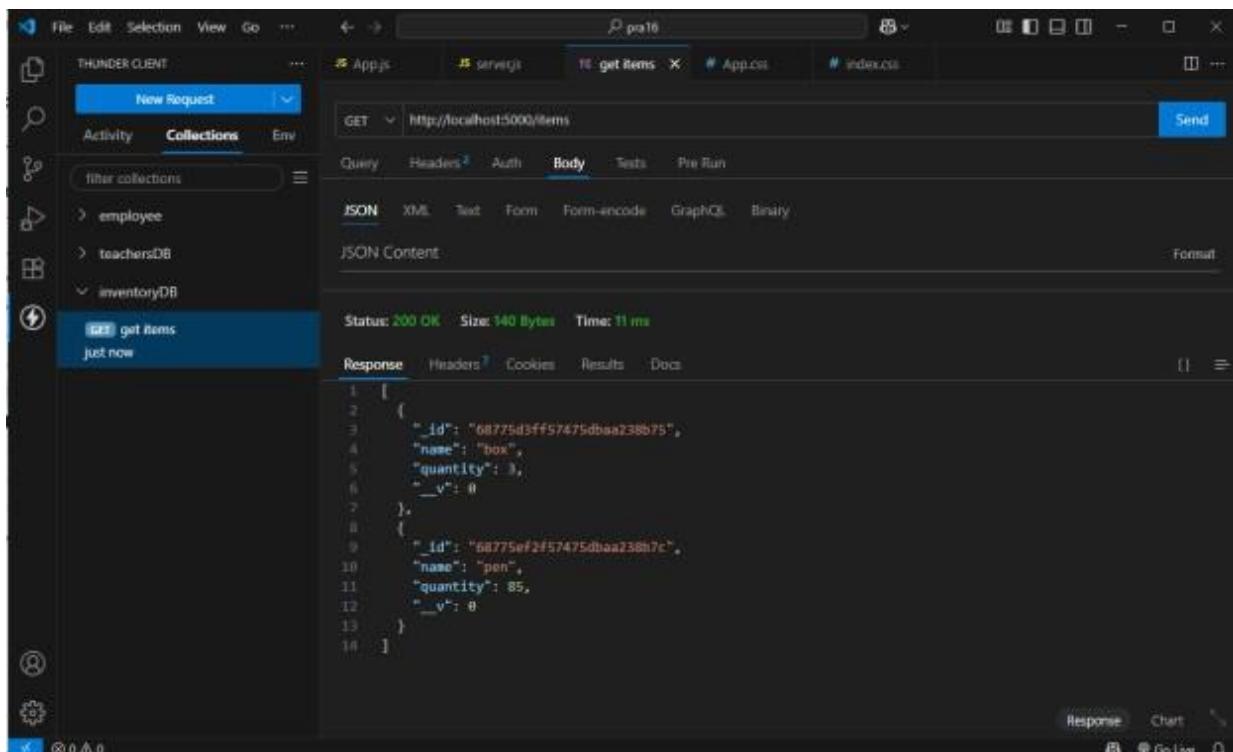
```
node server.js
```

Frontend:

```
cd inventory-mini-client
```

```
npm start
```

Output:



The screenshot shows the Thunder Client interface. On the left, there's a sidebar with 'THUNDER CLIENT' at the top, followed by tabs for 'Activity', 'Collections', and 'Env'. Under 'Collections', there are three items: 'employee', 'teachersDB', and 'inventoryDB'. 'inventoryDB' is expanded, showing a recent request named 'get items' made 'just now'. The main panel shows a request configuration for a 'GET' request to 'http://localhost:5000/items'. The 'Body' tab is selected, showing JSON content: 'Status: 200 OK', 'Size: 140 Bytes', and 'Time: 11 ms'. The 'Response' tab displays the JSON data returned from the server.

```
1 [
2   {
3     "_id": "68775d3ff57475dbaa238b75",
4     "name": "box",
5     "quantity": 3,
6     "__v": 0
7   },
8   {
9     "_id": "68775ef2f57475dbaa238b7c",
10    "name": "pen",
11    "quantity": 85,
12    "__v": 0
13  }
14]
```

The screenshot shows the Thunder Client interface with a successful POST request to `http://localhost:5000/items`. The request body contains:

```
1 {
2   "name": "Notebooks",
3   "quantity": 56
4 }
```

The response status is `200 OK`, size is `75 Bytes`, and time is `7 ms`. The response body is:

```
1 {
2   "name": "Notebooks",
3   "quantity": 56,
4   "id": "687760caf57475dbaa238b80",
5   "_v": 0
6 }
```

The screenshot shows the Thunder Client interface with a successful DELETE request to `http://localhost:5000/items/687760caf57475dbaa238b75`. The request body is empty.

The response status is `200 OK`, size is `7 Bytes`, and time is `7 ms`. The response body is:

```
1 Deleted
```



Inventory

Item Name Add

- box - Qty: 3
- pen - Qty: 85
- Notebooks - Qty: 56



Inventory

Item Name Add

- pen - Qty: 85
- Notebooks - Qty: 56

MongoDB Compass - localhost:27017/inventoryDB.items

Connections Edit View Collection Help

Compass

My Queries

localhost:27017 > inventoryDB > items

localhost:27017 > inventoryDB > items

Documents 2 Aggregations Schema Indexes Validation

Type a query: { field: "value" } or [generate query](#)

Explain Reset Find Options

ADD DATA EXPORT DATA UPDATE DELETE

26 1-2 of 2

`_id: ObjectId('5a9f775ef2f97475ebea2336fc')`
name: "pen"
quantity: 89
__v: 0

`_id: ObjectId('5a9f775ef2f97475ebea2336fb')`
name: "Notebooks"
quantity: 56
__v: 0

The screenshot shows the MongoDB Compass interface. On the left, the 'Connections' sidebar lists databases: 'localhost:27017' (selected), 'localhost:27017' (local), 'student', 'teachersDB', and 'localhost:27017'. The 'inventoryDB' database is expanded, showing collections: 'admin', 'config', 'InventoryDB' (selected), and 'Teachers'. The 'Items' collection is selected under 'InventoryDB'. The main panel displays the 'Documents' tab with two documents. Document 1 has fields '_id' (ObjectId('5a9f775ef2f97475ebea2336fc')), 'name' ('pen'), 'quantity' (89), and '__v' (0). Document 2 has fields '_id' (ObjectId('5a9f775ef2f97475ebea2336fb')), 'name' ('Notebooks'), 'quantity' (56), and '__v' (0). Below the documents are buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. At the bottom, there are pagination controls (26, 1-2 of 2) and a toolbar with 'Find' and 'Options'.

