**Program: 1. Demonstrate Node .Js Application to perform CRUD operation for online Book Cart**

**Prelims which is needed before running the programs:**

**$npm init -y**

**$npm install express**

**Example Program:**

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

**const app = express();**

**const port = 3000;**

**// Middleware to parse JSON bodies**

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

**// Our in-memory JSON data store**

**let books = [**

**{ id: 1, title: 'The Lord of the Rings', author: 'J.R.R. Tolkien', price: 25.00, quantity: 10 },**

**{ id: 2, title: 'Pride and Prejudice', author: 'Jane Austen', price: 15.50, quantity: 5 },**

**{ id: 3, title: 'To Kill a Mockingbird', author: 'Harper Lee', price: 10.00, quantity: 7 }**

**];**

**app.get('/books', (req, res) => {**

**res.status(200).json(books);**

**});**

**app.get('/books/:id', (req, res) => {**

**const book = books.find((b) => b.id === parseInt(req.params.id));**

**console.log(typeof(req.params.id)," ",book);**

**if(book){**

**res.status(200).json(book);**

**}else{**

**res.status(201).send(" book not found");**

**}**

**});**

**app.post('/books', (req, res) => {**

**const newBook = {**

**id : books.length + 1,**

**title:req.body.title,**

**author:req.body.author,**

**price:req.body.price,**

**quantity:req.body.quantity**

**};**

**books.push(newBook);**

**res.status(201).json(newBook);**

**});**

**app.patch('/books',(req,res,err)=>{**

**res.send("in the patch method");**

**});**

**app.put('/books',(req,res,err)=>{**

**res.send("in the put method");**

**});**

**app.delete('/books/:id', (req, res) => {**

**const initialLength = books.length;**

**books = books.filter(book => book.id !== parseInt(req.params.id));**

**if (books.length === initialLength) {**

**return res.status(404).json({ message: 'Book not found' });**

**}**

**res.status(200).json({ message: 'Book deleted successfully' });**

**});**

**app.listen(port, () => {**

**console.log(`Server is running on http://localhost:${port}`);**

**});**

**Expected OUTPUT:**

**Program 2: Write a node.js program using Express framework to accept user name, Branch, Semester, from web page and display the information as below**

**a) Handle both get and post methods**

**b) Branch should be underlined**

**c) Name should be in bold face.**

**Prelims which is needed before running the programs:**

**$npm init -y**

**$npm install express**

**Program: index.html**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>Student Information</title>**

**<style>**

**body { font-family: Arial, sans-serif; padding: 20px; }**

**.container { max-width: 500px; margin: auto; }**

**h2 { text-align: center; }**

**form { margin-bottom: 20px; border: 1px solid #ccc; padding: 20px; border-radius: 8px; }**

**label { display: block; margin-bottom: 5px; }**

**input[type="text"] { width: 100%; padding: 8px; margin-bottom: 10px; box-sizing: border-box; }**

**button { padding: 10px 15px; cursor: pointer; }**

**.output { margin-top: 20px; padding: 15px; border: 1px solid #007BFF; background-color: #e6f2ff; border-radius: 8px; }**

**</style>**

**</head>**

**<body>**

**<div class="container">**

**<h2>Student Information Form</h2>**

**<form action="/submit-get" method="get">**

**<h3>Using GET Method</h3>**

**<label for="name-get">Name:</label>**

**<input type="text" id="name-get" name="name" required>**

**<label for="branch-get">Branch:</label>**

**<input type="text" id="branch-get" name="branch" required>**

**<label for="semester-get">Semester:</label>**

**<input type="text" id="semester-get" name="semester" required>**

**<button type="submit">Submit (GET)</button>**

**</form>**

**<form action="/submit-post" method="post">**

**<h3>Using POST Method</h3>**

**<label for="name-post">Name:</label>**

**<input type="text" id="name-post" name="name" required>**

**<label for="branch-post">Branch:</label>**

**<input type="text" id="branch-post" name="branch" required>**

**<label for="semester-post">Semester:</label>**

**<input type="text" id="semester-post" name="semester" required>**

**<button type="submit">Submit (POST)</button>**

**</form>**

**</div>**

**</body>**

**</html>**

**Example Program: server.js**

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

**const path = require('path');**

**const app = express();**

**const port = 3000;**

**// Middleware to serve static files (like your HTML file)**

**app.use(express.static(path.join(\_\_dirname)));**

**// Middleware to parse URL-encoded data (for POST requests)**

**app.use(express.urlencoded({ extended: true }));**

**// GET method handler**

**app.get('/submit-get', (req, res) => {**

**// Data is in req.query for GET requests**

**const name = req.query.name;**

**const branch = req.query.branch;**

**const semester = req.query.semester;**

**// Display the information with specified formatting**

**const htmlResponse = ` <h2>Student Information (GET)</h2>**

**<p>Name: <b>${name}</b></p>**

**<p>Branch: <u>${branch}</u></p>**

**<p>Semester: ${semester}</p>**

**<br> <a href="/">Go Back</a>`;**

**res.send(htmlResponse);**

**});**

**// POST method handler**

**app.post('/submit-post', (req, res) => {**

**// Data is in req.body for POST requests**

**const name = req.body.name;**

**const branch = req.body.branch;**

**const semester = req.body.semester;**

**// Display the information with specified formatting**

**const htmlResponse = `**

**<h2>Student Information (POST)</h2>**

**<p>Name: <b>${name}</b></p>**

**<p>Branch: <u>${branch}</u></p>**

**<p>Semester: ${semester}</p>**

**<br>**

**<a href="/">Go Back</a>**

**`;**

**res.send(htmlResponse);**

**});**

**// Start the server**

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

**console.log(`Server is listening at http://localhost:${port}`);**

**console.log('Open your browser and navigate to http://localhost:3000/index.html');**

**});**

**$node server**

**go to this URL  -->   http://localhost:3000/index.html**

**Expected OUTPUT**

**Design a resume of a job aspirant using React components like Classes and Functions. Style the resume by applying CSS**

**Prelims which is needed before running the programs:**

**$npx create-react-app program3**

**Example Program: /src/ResumeCreate/Education.js**

**import React from "react";**

**class EdTable extends React.Component {**

**render() {**

**const education = [**

**{**

**id: 1,**

**name: "National Public School",**

**place: "Bengaluru",**

**stddeg: "10th",**

**grade: "A+",**

**},**

**{**

**id: 2,**

**name: "Indian Institute of Science",**

**place: "Bengaluru",**

**stddeg: "under grade",**

**grade: "9.8",**

**},**

**]**

**const edudetails = education.map(educ =>**

**<EdRow id={educ.id}**

**name={educ.name}**

**place={educ.place}**

**stddeg={educ.stddeg}**

**grade={educ.grade} /> );**

**return (**

**<table>**

**<tr>**

**<th>id</th>**

**<th>name</th>**

**<th>place</th>**

**<th>education</th>**

**<th>grade</th>**

**</tr>**

**<tbody>**

**{edudetails}**

**</tbody>**

**</table>**

**);**

**}**

**}**

**class EdRow extends React.Component {**

**render() {**

**return (**

**<tr>**

**<td>{this.props.id}</td>**

**<td>{this.props.name}</td>**

**<td>{this.props.place}</td>**

**<td>{this.props.stddeg}</td>**

**<td>{this.props.grade}</td>**

**</tr>**

**);**

**}**

**}**

**class Exducation extends React.Component {**

**render() {**

**return (**

**<section>**

**<EdTable />**

**</section>**

**);**

**}**

**}**

**export default Education;**

**Example Program: Resume\_Header.js**

**import React from "react";**

**class Resume\_Header extends React.Component{**

**render(){**

**return(**

**<h1>Resume Details</h1>**

**);**

**}**

**}**

**export default Resume\_Header;**

**Example Program: Resume\_Person\_Info.js**

**import React from "react";**

**class Resume\_Person\_Info extends React.Component {**

**render() {**

**return (**

**<div >**

**<div > Name : <p>Prashanth K</p></div><br/>**

**<div >Address : <p>Kengeri</p></div><br/>**

**<div >Phone No : <p>1234567890</p></div><br/>**

**<div >email id : <p>prashanthk@rvce.edu.in</p></div>**

**</div>**

**);**

**}**

**}**

**export default Resume\_Person\_Info;**

**Example Program: Index.js**

**import React from 'react';**

**import ReactDOM from 'react-dom/client';**

**import './index.css';**

**import Resume\_Header from './ResumeCreate/Resume\_Header';**

**import Resume\_Person\_Info from './ResumeCreate/Resume\_Person\_Info';**

**import Education from './ResumeCreate/Educational';**

**const root = ReactDOM.createRoot(document.getElementById('root'));**

**root.render(**

**<React.StrictMode>**

**<Resume\_Header />**

**<Resume\_Person\_Info />**

**<Education />**

**</React.StrictMode>**

**);**

**Steps for execution:**

**$npm start**

**> prog3@0.1.0 start**

**> react-scripts start**

**Expected OUTPUT**

**Program 4: Build student registration portal using Entities like component, state and props**

**Prelims which is needed before running the programs:**

**$npx create-react-app program4**

**Example Program: /src/Student.js**

**import React from "react";**

**const studlist = [{**

**slno: 1,**

**name: "stud1",**

**usn: "rvce1",**

**tmarks: 150,**

**}, {**

**slno: 2,**

**name: "stud2",**

**usn: "rvce2",**

**tmarks: 145,**

**}]**

**class StudentRow extends React.Component {**

**render() {**

**const studl = this.props.stdli;**

**return (**

**<tr>**

**<th>{studl.slno}</th>**

**<th>{studl.name}</th>**

**<th>{studl.usn}</th>**

**<th>{studl.tmarks}</th>**

**</tr>**

**);**

**}**

**}**

**class StudentsTable extends React.Component {**

**render() {**

**const studrows = this.props.studlist.map(studli =>**

**<StudentRow key={studli.slno} stdli={studli} />**

**);**

**return (**

**<table>**

**<tr>**

**<th>Sl No</th>**

**<th>Student Name</th>**

**<th>USN</th>**

**<th>Total Marks</th>**

**</tr>**

**<tbody>**

**{studrows}**

**</tbody>**

**</table>**

**);**

**}**

**}**

**class StudentAddSubmit extends React.Component {**

**constructor() {**

**super();**

**this.handleSubmit = this.handleSubmit.bind(this);**

**}**

**handleSubmit(e) {**

**e.preventDefault();**

**const form = document.forms.studAdd;**

**const studli = {**

**name: form.name.value,**

**usn: form.usn.value,**

**tmarks: form.tmarks.value,**

**}**

**console.log("studli - ",studli);**

**this.props.CreateStudentAdd(studli);**

**form.name.value = ""; form.usn.value = ""; form.tmarks.value = ""**

**}**

**render() {**

**return (**

**<form name="studAdd" onSubmit={this.handleSubmit}>**

**<input type="text" name="name" placeholder="Name" />**

**<input type="text" name="usn" placeholder="USN" />**

**<input type="text" name="tmarks" placeholder="Total Marks" />**

**<button>Add</button>**

**</form>**

**);**

**}**

**}**

**class StudentListTable extends React.Component{**

**constructor() {**

**super();**

**this.state = { newStudList: [] };**

**this.CreateStudentAdd = this.CreateStudentAdd.bind(this);**

**}**

**componentDidMount() {**

**this.loadData();**

**}**

**loadData() {**

**setTimeout(() => {**

**this.setState({ newStudList: studlist })**

**}, 5000);**

**console.log("i have loaded the student add component also");**

**}**

**CreateStudentAdd(studli) {**

**const studlength = this.state.newStudList.length+1;**

**studli.slno = studlength;**

**const newstudlist1 = this.state.newStudList.slice();**

**newstudlist1.push(studli);**

**this.setState({ newStudList: newstudlist1 });**

**}**

**render(){**

**return(**

**<React.Fragment>**

**<hr/>**

**<StudentsTable studlist = {this.state.newStudList}/>**

**<hr/>**

**<StudentAddSubmit CreateStudentAdd={this.CreateStudentAdd}/>**

**</React.Fragment>**

**);**

**}**

**}**

**export default StudentListTable;**

**Steps for execution:**

**$npm start**

**> prog3@0.1.0 start**

**> react-scripts start**

**Expected OUTPUT**

**Design and implement a React Form that collects user input for name, email, and password. Validate the form using Regular Expression.**

**Prelims which is needed before running the programs:**

**$npx create-react-app program5**

**Example Program: /src/RegistrationForm.js**

**import React from "react";**

**class RegistrationForm extends React.Component {**

**state = {**

**name: "",**

**email: "",**

**password: "",**

**errors: {}**

**};**

**validate() {**

**const errors = {};**

**if (!this.state.name) {**

**errors.name = "Name is required";**

**} else if (this.state.name.length < 3) {**

**errors.name = "Name must be at least 3 letters";**

**}**

**if (!this.state.email) {**

**errors.email = "Email is required";**

**} else if (!/\S+@\S+\.\S+/.test(this.state.email)) {**

**errors.email = "Email is invalid";**

**}**

**if (!this.state.password) {**

**errors.password = "Password is required";**

**} else if (this.state.password.length < 8) {**

**errors.password = "Password must be at least 8 letters";**

**}**

**this.setState({ errors });**

**return Object.keys(errors).length === 0;**

**};**

**handleChange(e){**

**this.setState({ [e.target.name]: e.target.value });**

**};**

**handleSubmit(e){**

**e.preventDefault();**

**if (this.validate()) {**

**alert("Form submitted!");**

**// Reset form or further action**

**this.setState({ name: "", email: "", password: "", errors: {} });**

**}**

**};**

**render() {**

**const { name, email, password, errors } = this.state;**

**return (**

**<div><h1>Registration form for USER Input:</h1>**

**<form onSubmit={this.handleSubmit}>**

**<input**

**name="name"**

**placeholder="Name"**

**value={name}**

**onChange={this.handleChange}**

**/>**

**<div>{errors.name}</div>**

**<input**

**name="email"**

**placeholder="Email"**

**value={email}**

**onChange={this.handleChange}**

**/>**

**<div>{errors.email}</div>**

**<input**

**name="password"**

**type="password"**

**placeholder="Password"**

**value={password}**

**onChange={this.handleChange}**

**/>**

**<div>{errors.password}</div>**

**<button type="submit">Register</button>**

**</form>**

**</div>**

**);**

**}**

**}**

**export default RegistrationForm;**

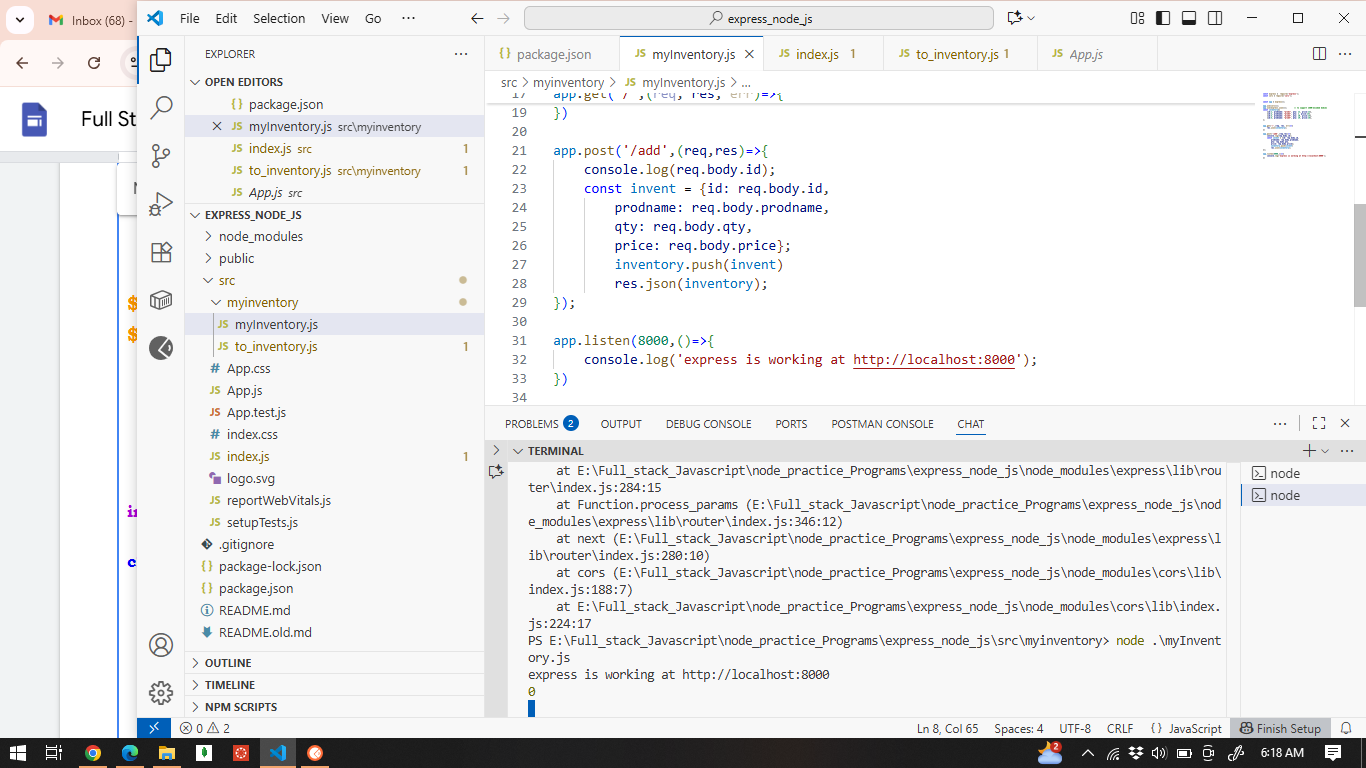
**Steps for execution:**

**$npm start**

**> prog3@0.1.0 start**

**> react-scripts start**

**Expected OUTPUT**



**Deploy connectivity between React and Node Application for Inventory Management system**

**Prelims which is needed before running the programs:**

**$npx create-react-app program6**

**$npm install express cors axios**

**Example Program: myinventory.js**

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

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

**const app = express();**

**app.use(cors());**

**app.use(express.json());       // to support JSON-encoded bodies**

**const inventory=[**

**{id:1, prodname: "prod1", qty: 12, price:12},**

**{id:2, prodname: "prod2", qty: 1, price:13},**

**{id:3, prodname: "prod3", qty: 10, price:14},**

**{id:4, prodname: "prod4", qty: 19, price:16},**

**];**

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

**res.json(inventory);**

**})**

**app.post('/add',(req,res)=>{**

**console.log(req.body.id);**

**const invent = {id: req.body.id,**

**prodname: req.body.prodname,**

**qty: req.body.qty,**

**price: req.body.price};**

**inventory.push(invent)**

**res.json(inventory);**

**});**

**app.listen(8000,()=>{**

**console.log('express is working at http://localhost:8000');**

**})**

**Steps for execution:**

**$node myinventory.js**

**Example Program: to\_inventory.js**

**import React, { useState } from "react";**

**import axios from "axios";**

**function Display\_Inventory() {**

**const [res, setres] = useState([]);**

**const resp = async () => {**

**const response = await axios.get("http://localhost:8000");**

**console.log("asdfasdf", response.data)**

**setres(response.data);**

**}**

**resp();**

**return (**

**<div>**

**<h1>Inventory Management</h1>**

**<table>**

**<thead>**

**<tr>**

**<th>ID</th>**

**<th>Product Name</th>**

**<th>Quantity</th>**

**<th>Price</th>**

**</tr>**

**</thead>**

**<tbody>**

**{res.map((inventory) => (**

**<tr key={inventory.id}>**

**<td>{inventory.id}</td>**

**<td>{inventory.name}</td>**

**<td>{inventory.qty}</td>**

**<td>{inventory.price}</td>**

**</tr>**

**))}**

**</tbody>**

**</table>**

**</div>**

**);**

**}**

**function AddInventory() {**

**const [ id, setId ] = useState(0);**

**const [ name, setName ] = useState('');**

**const [ qty, setQty ] = useState(0);**

**const [ price, setPrice ] = useState(0);**

**const [inventory, setInventory] =useState([]);**

**const SubmitEvent = () => {**

**const fo = { "id":id,"name": name,"qty":qty, "price":price };**

**console.log("fo ",fo);**

**const resp = async () => {**

**const response = await axios.post("http://localhost:8000/add", fo);**

**console.log(response.data)**

**const getresponse = await axios.get("http://localhost:8000");**

**setInventory(getresponse.data);**

**}**

**resp();**

**}**

**return (**

**<div>**

**<h1>Inventory Management1</h1>**

**<table>**

**<thead>**

**<tr>**

**<th>ID</th>**

**<th>Product Name</th>**

**<th>Quantity</th>**

**<th>Price</th>**

**</tr>**

**</thead>**

**<tbody>**

**<tr>**

**<td>id</td><td><input type="number"**

**name="id"**

**value={id}**

**onChange={(e) => setId(e.target.value)}**

**/></td></tr>**

**<tr><td>Product Name</td><td><input type="text"**

**name="prodname"**

**value={name}**

**onChange={(e) => setName(e.target.value)} /></td>**

**</tr>**

**<tr><td>Quantity</td><td><input type="number"**

**name="qty"**

**value={qty}**

**onChange={(e) => setQty(e.target.value)} /></td>**

**</tr>**

**<tr><td>Price</td><td><input type="number"**

**name="price"**

**value={price}**

**onChange={(e) => setPrice(e.target.value)} /></td></tr>**

**<tr><td><input type="submit" name="onSubmit" onClick={SubmitEvent} /></td></tr>**

**</tbody>**

**</table>**

**</div>**

**);**

**}**

**export { Display\_Inventory, AddInventory };**

**index.js**

**import React from 'react';**

**import ReactDOM from 'react-dom/client';**

**import './index.css';**

**import { AddInventory, Display\_Inventory } from './to\_inventory';**

**const root = ReactDOM.createRoot(document.getElementById('root'));**

**root.render(**

**<React.StrictMode>**

**<Display\_Inventory/>**

**<AddInventory />**

**</React.StrictMode>**

**);**

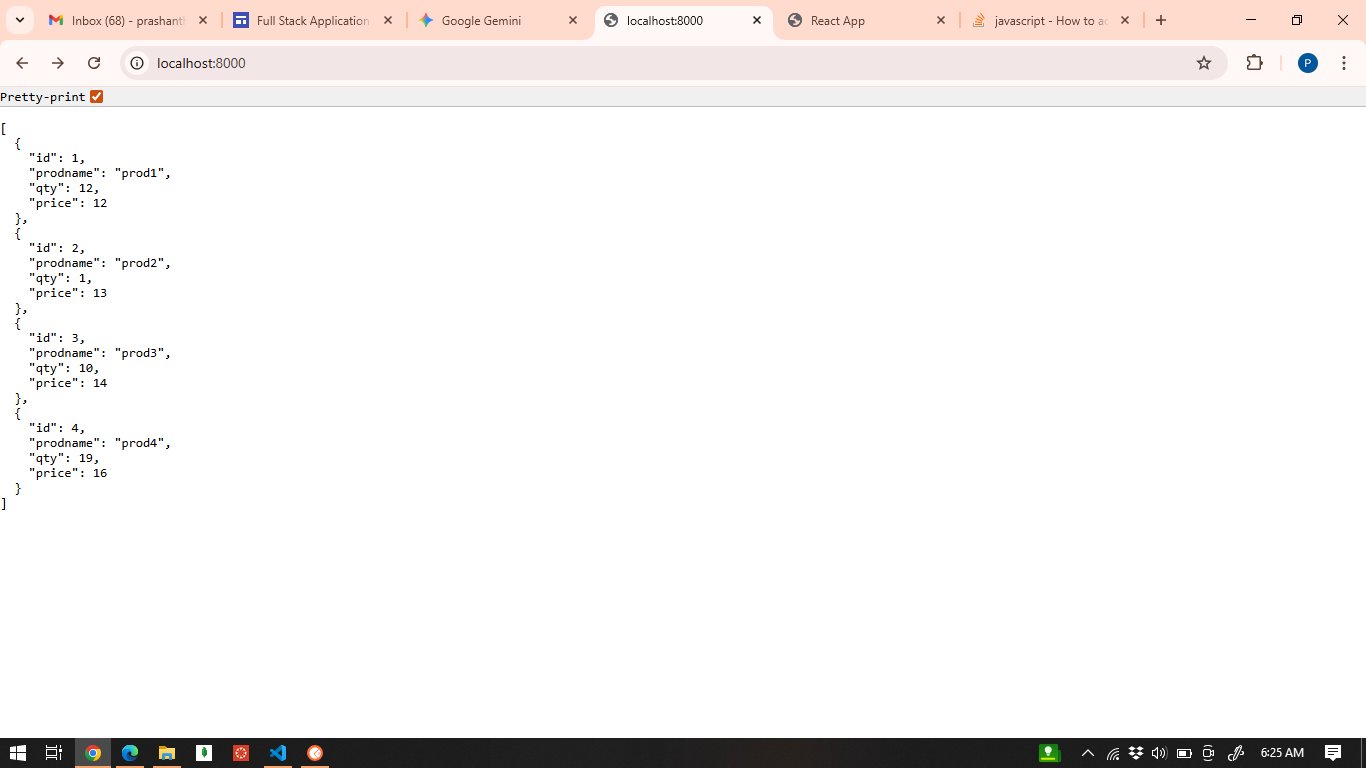
**Execution Procedure:**

**$npm start**

**> prog6@0.1.0 start**

**> react-scripts start**

**Expected OUTPUT**



**Program 7: Develop a MongoDB query for comparison selectors, Logical Selectors for Company database**

**$mongosh**

**## Setup: Sample employees Collection 🏢**

First, let's insert some sample data into an employees collection to use for our queries.

**db.employees.insertMany([**

**{**

**employee\_id: 101,**

**name: "Sonia Rao",**

**department: "Engineering",**

**salary: 95000,**

**hire\_date: new Date("2023-01-15"),**

**performance\_score: 4.5,**

**status: "Active"**

**},   {**

**employee\_id: 102,**

**name: "Rajesh Kumar",**

**department: "HR",**

**salary: 68000,**

**hire\_date: new Date("2024-03-22"),**

**performance\_score: 4.1,**

**status: "Active"**

**},   {**

**employee\_id: 103,**

**name: "Arjun Singh",**

**department: "Engineering",**

**salary: 120000,**

**hire\_date: new Date("2022-06-01"),**

**performance\_score: 4.8,**

**status: "Active"**

**},   {**

**employee\_id: 104,**

**name: "Priya Mehta",**

**department: "Sales",**

**salary: 85000,**

**hire\_date: new Date("2023-11-10"),**

**performance\_score: 4.2,**

**status: "Active"**

**},   {**

**employee\_id: 105,**

**name: "Vikram Reddy",**

**department: "Sales",**

**salary: 85000,**

**hire\_date: new Date("2024-08-01"),**

**status: "On Leave"**

**}**

**]);**

**## 1. Comparison Selectors ⚙️**

These operators compare the value of a field against a specified value.

**$gt (Greater Than) & $lte (Less Than or Equal To)**

* **Goal:** Find all employees with a **salary** greater than $90,000 but less than or equal to $120,000.
* **Query:**

**db.employees.find({**

**salary: { $gt: 90000, $lte: 120000 }**

**})**

* **Explanation:** This query finds documents where the **salary** is between 90,001 and 120,000, inclusive. It will find Sonia Rao and Arjun Singh.

**$in (In an Array)**

* **Goal:** Find all employees who work in either the "HR" or "Sales" **department**.
* **Query:**

**db.employees.find({**

**department: { $in: ["HR", "Sales"] }**

**})**

* **Explanation:** The **$in** operator selects documents where the **department** field's value exactly matches any value in the provided array. It will find Rajesh Kumar, Priya Mehta, and Vikram Reddy.

**$ne (Not Equal)**

* **Goal:** Find all employees whose **status** is not "On Leave".
* **Query:**

**db.employees.find({**

**status: { $ne: "On Leave" }**

**})**

* **Explanation:** The **$ne** operator selects all documents where the **status** field value is not equal to the specified value.

**## 2. Logical Selectors**

These operators are used to combine multiple query conditions.

**$and (Implicit)**

* **Goal:** Find employees in the "Engineering" **department** with a **performance\_score** of 4.5 or higher.
* **Query:**

**db.employees.find({**

**department: "Engineering",**

**performance\_score: { $gte: 4.5 }**

**})**

* **Explanation:** When you list multiple fields separated by a comma in a find() query, MongoDB treats it as an **$and** condition by default. This is the most common way to combine conditions. This query will find Sonia Rao and Arjun Singh.

**$or**

* **Goal:** Find employees who are either in the "HR" **department** OR have a **salary** greater than $100,000.
* **Query:**

**db.employees.find({**

**$or: [**

**{ department: "HR" },**

**{ salary: { $gt: 100000 } }**

**]**

**})**

* **Explanation:** The **$or** operator takes an array of condition objects and selects documents that satisfy *at least one* of the conditions. This will find Rajesh Kumar (matches department) and Arjun Singh (matches salary).

**## 3. Combining Selectors for a Complex Query ✨**

Real-world scenarios often require combining multiple operators.

* **Goal:** Find all **Active** employees in either the **Engineering** or **Sales** departments who were hired **before 2024**.
* **Query:**

**db.employees.find({**

**status: "Active",**

**department: { $in: ["Engineering", "Sales"] },**

**hire\_date: { $lt: new Date("2024-01-01") }**

**})**

**Explanation:** This query implicitly uses $and to combine three conditions:

1. The **status** must be "Active".
2. The **department** must be one of the values in the $in array
3. The **hire\_date** must be less than ($lt) the specified date.

**Program 8: Execute aggregation pipeline and its operation to illustrate text search on catalog data collection**

**Prelims which is needed before running the programs:**

**$mongosh**

**## Step 1: Prepare Your Data Collection**

First, you need a collection with text-rich fields to search. We'll use a catalog collection for this example. The key is to have fields like name and description that contain the words you want to find.

**Action:** In mongosh, connect to your database and run the following command to insert sample data.

**db.catalog.insertMany([**

**{**

**item\_id: "A001",**

**name: "Classic Leather Backpack",**

**description: "Durable leather backpack with multiple compartments, perfect for daily commute or light travel.",**

**tags: ["leather", "backpack", "travel", "work"],**

**price: 120.00,**

**in\_stock: true**

**},**

**{**

**item\_id: "A002",**

**name: "Ergonomic Office Chair",**

**description: "Adjustable office chair with lumbar support and breathable mesh, ideal for long working hours.",**

**tags: ["office", "chair", "ergonomic", "comfort"],**

**price: 280.00,**

**in\_stock: false**

**},**

**{**

**item\_id: "A003",**

**name: "Durable Cotton T-Shirt",**

**description: "A soft and durable t-shirt made from 100% organic cotton. Eco-friendly and comfortable.",**

**tags: ["cotton", "tshirt", "organic", "eco-friendly"],**

**price: 25.00,**

**in\_stock: true**

**}**

**]);**

**## Step 2: Create a text Index**

This is the most critical step for enabling text search on a Community Server. You must tell MongoDB which fields to prepare for searching. Without this index, the $text operator will not work.

**Why is this needed?** A text index acts like an index at the back of a book. It lists all the significant words from your specified fields and maps them back to the documents they came from. This allows MongoDB to find words very quickly without scanning every single document in the collection.

**Important Note:** A collection can have **at most one** text index.

**Action:** Run the createIndex command. Here, we are indexing the name, description, and tags fields.

**db.catalog.createIndex({**

**name: "text",    description: "text",    tags: "text"**

**})**

**## Step 3: Build and Execute the Aggregation Pipeline**

Now, you'll construct the query. Instead of the $search stage used in Atlas, you will use the **$match** stage combined with the **$text** operator.

**Action:** Let's build a pipeline to find all items that contain the words "durable" or "backpack".

1. **Stage 1: The $match Stage with $text** This stage filters the documents.
   * **$text**: The operator that performs the search on the text index.
   * **$search**: The string containing the words you're looking for. By default, MongoDB searches for documents that contain *any* of the space-separated words (an OR condition).

**{**

**$match: {**

**$text: {**

**$search: "durable backpack"**

**}**

**}**

**}**

**Stage 2: The $project Stage** This stage refines the output. It's good practice to select only the fields you need and to view the search relevance score.

* **$meta: "textScore"**: This special expression retrieves the relevance score calculated by the $text search. A higher score means a better match.

**{**

**$project: {**

**\_id: 0,**

**name: 1,**

**price: 1,**

**score: { $meta: "textScore" }**

**}**

**}**

**Complete Pipeline Execution:** Combine these stages and run the aggregate command.

db.catalog.aggregate([

  // Stage 1: Filter documents using the text index

  {

    $match: {

      $text: {

        $search: "durable backpack"

      }

    }

  },

  // Stage 2: Reshape output and include the relevance score

  {

    $project: {

      \_id: 0,

      name: 1,

      price: 1,

      score: { $meta: "textScore" }

    }

  }

])

**## Step 4: Analyze the Results**

The output will be the documents that matched your search query, formatted according to your $project stage.

**Expected Output:**

**[**

**{**

**"name": "Classic Leather Backpack",**

**"price": 120,**

**"score": 1.5**

**},**

**{**

**"name": "Durable Cotton T-Shirt",**

**"price": 25,**

**"score": 0.75**

**}**

**]**

**Program 9: Design an employee Management system using RESTFULL APIs in React**

**Prelims which is needed before running the programs:**

**$npx create-react-app program9**

**$npm install axios**

**db.json**

**{**

**"employees": [**

**{ "id": 1, "firstName": "John", "lastName": "Doe", "email": "john.doe@example.com" },**

**{ "id": 2, "firstName": "Jane", "lastName": "Smith", "email": "jane.smith@example.com" }**

**]**

**}**

**$npx json-server --watch db.json --port 5000**

**Example - /src/**[**App.js**](http://App.js)

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

**import axios from 'axios';**

**import './App.css'; // For basic styling**

**// API URL**

**const API\_URL = "http://localhost:5000/employees";**

**function App() {**

**// State to hold the list of employees**

**const [employees, setEmployees] = useState([]);**

**// State for the form data. 'null' if we are not editing/adding.**

**const [currentEmployee, setCurrentEmployee] = useState(null);**

**// Fetch employees when the component loads**

**useEffect(() => {**

**fetchEmployees();**

**}, []);**

**// R: Read employees from the API**

**const fetchEmployees = async () => {**

**const response = await axios.get(API\_URL);**

**setEmployees(response.data);**

**};**

**// C & U: Handle form submission for Create and Update**

**const handleSave = async (employee) => {**

**if (employee.id) {**

**// Update existing employee**

**await axios.put(`${API\_URL}/${employee.id}`, employee);**

**} else {**

**// Create new employee**

**await axios.post(API\_URL, employee);**

**}**

**fetchEmployees(); // Refresh the list**

**setCurrentEmployee(null); // Hide the form**

**};**

**// D: Delete an employee**

**const handleDelete = async (id) => {**

**await axios.delete(`${API\_URL}/${id}`);**

**fetchEmployees(); // Refresh the list**

**};**

**// Handlers to show the form**

**const handleAdd = () => {**

**setCurrentEmployee({ firstName: '', lastName: '', email: '' }); // Show form with empty fields**

**};**

**const handleEdit = (employee) => {**

**setCurrentEmployee(employee); // Show form with pre-filled data**

**};**

**// If 'currentEmployee' is not null, show the form. Otherwise, show the list.**

**if (currentEmployee) {**

**return <EmployeeForm employee={currentEmployee} onSave={handleSave} onCancel={() => setCurrentEmployee(null)} />;**

**}**

**return (**

**<div className="container">**

**<h2>Employees List</h2>**

**<button onClick={handleAdd}>Add Employee</button>**

**<table>**

**<thead>**

**<tr>**

**<th>First Name</th>**

**<th>Last Name</th>**

**<th>Email</th>**

**<th>Actions</th>**

**</tr>**

**</thead>**

**<tbody>**

**{employees.map(emp => (**

**<tr key={emp.id}>**

**<td>{emp.firstName}</td>**

**<td>{emp.lastName}</td>**

**<td>{emp.email}</td>**

**<td>**

**<button onClick={() => handleEdit(emp)}>Edit</button>**

**<button onClick={() => handleDelete(emp.id)}>Delete</button>**

**</td>**

**</tr>**

**))}**

**</tbody>**

**</table>**

**</div>**

**);**

**}**

**// A simple form component defined in the same file**

**const EmployeeForm = ({ employee, onSave, onCancel }) => {**

**const [formData, setFormData] = useState({ ...employee });**

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

**e.preventDefault();**

**onSave(formData);**

**};**

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

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

**setFormData(prev => ({ ...prev, [name]: value }));**

**};**

**return (**

**<form onSubmit={handleSubmit}>**

**<h2>{formData.id ? 'Edit Employee' : 'Add Employee'}</h2>**

**<input name="firstName" value={formData.firstName} onChange={handleChange} placeholder="First Name" required />**

**<input name="lastName" value={formData.lastName} onChange={handleChange} placeholder="Last Name" required />**

**<input name="email" value={formData.email} onChange={handleChange} placeholder="Email" required />**

**<button type="submit">Save</button>**

**<button type="button" onClick={onCancel}>Cancel</button>**

**</form>**

**);**

**};**

**export default App;**

**Expected Output:**

**Program 10: Create a React application using react-router-dom with multiple pages (Home, About, Contact).**

**Prelims which is needed before running the programs:**

**$npx create-react-app program10**

**$npm install react-router-dom axios cors**

**Example Program: src/pages/Home.js**

**import React from 'react';**

**function Home() {**

**return (**

**<div>**

**<h1>Home Page</h1>**

**<p>Welcome to the homepage of our application!</p>**

**</div>**

**);**

**}**

**export default Home;**

**Example Program: src/pages/About**[**.**](http://home.js)**js**

**import React from 'react';**

**function About() {**

**return (**

**<div>**

**<h1>About Us</h1>**

**<p>This is the about page, where you can learn more about us.</p>**

**</div>**

**);**

**}**

**export default About;**

**Example Program: src/pages/Contact**[**.**](http://home.js)**js**

**import React from 'react';**

**function Contact() {**

**return (**

**<div>**

**<h1>Contact Page</h1>**

**<p>Get in touch with us through this contact page.</p>**

**</div>**

**);**

**}**

**export default Contact;**

**Example Program: src/App.js**

**import React from 'react';**

**import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';**

**import Home from './pages/Home';**

**import About from './pages/About';**

**import Contact from './pages/Contact';**

**import './App.css';**

**function App() {**

**return (**

**<BrowserRouter>**

**<div className="App">**

**<nav>**

**<ul>**

**<li>**

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

**</li>**

**<li>**

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

**</li>**

**<li>**

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

**</li>**

**</ul>**

**</nav>**

**<main>**

**<Routes>**

**<Route path="/" element={<Home />} />**

**<Route path="/about" element={<About />} />**

**<Route path="/contact" element={<Contact />} />**

**</Routes>**

**</main>**

**</div>**

**</BrowserRouter>**

**);**

**}**

**export default App;**

**Execution Procedure:**

**$npm start**

**> prog6@0.1.0 start**

**> react-scripts start**

**Expected OUTPUT**