Sl.NO	Experiments
1.	a. Write a script that Logs "Hello, World!" to the console. Create a script that calculates the sum of two numbers and displays the result in an alert box. b. Create an array of 5 cities and perform the following operations: Log the total number of cities. Add a new city at the end. Remove the first city. Find and log the index of a specific city.
2.	a. Read a string from the user, Find its length. Extract the word "JavaScript" using substring() or slice(). Replace one word with another word and log the new string. Write a function isPalindrome(str) that checks if a given string is a palindrome (reads the same backward).
3.	Create an object student with properties: name (string), grade (number), subjects (array), displayInfo() (method to log the student's details) Write a script to dynamically add a passed property to the student object, with a value of true or false based on their grade. Create a loop to log all keys and values of the student object.
4.	Create a button in your HTML with the text "Click Me". Add an event listener to log "Button clicked!" to the console when the button is clicked. Select an image and add a mouseover event listener to change its border color. Add an event listener to the document that logs the key pressed by the user.
5.	Build a React application to track issues. Display a list of issues (use static data). Each issue should have a title, description, and status (e.g., Open/Closed). Render the list using a functional component.
6.	Create a component Counter with A state variable count initialized to 0. Create Buttons to increment and decrement the count. Simulate fetching initial data for the Counter component using useEffect (functional component) or componentDidMount (class component). Extend the Counter component to Double the count value when a button is clicked. Reset the count to 0 using another button.
7.	Install Express (npm install express). Set up a basic server that responds with "Hello, Express!" at the root endpoint (GET /). Create a REST API. Implement endpoints for a Product resource: GET : Returns a list of products. POST : Adds a new product. GET /:id: Returns details of a specific product. PUT /:id: Updates an existing product. DELETE /:id: Deletes a product. Add middleware to log requests to the console. Use express.json() to parse incoming JSON payloads.
8.	Install the MongoDB driver for Node.js. Create a Node.js script to connect to the shop database. Implement insert, find, update, and delete operations using the Node.js MongoDB driver. Define a product schema using Mongoose. Insert data into the products collection using Mongoose. Create an Express API with a /products endpoint to fetch all products. Use fetch in React to call the /products endpoint and display the list of products. Add a POST /products endpoint in Express to insert a new product. Update the Product List, After adding a product, update the list of products displayed in React.

1. a. Write a script that Logs "Hello, World!" to the console. Create a script that calculates the sum of two numbers and displays the result in an alert box.

b. Create an array of 5 cities and perform the following operations: Log the total number of cities. Add a new city at the end. Remove the first city. Find and log the index of a specific city.

A.

```
// Log "Hello, World!" to the console

console.log("Hello, World!");


// Function to calculate the sum of two numbers and display the result in an alert box

function calculateSum(num1, num2) {

    const sum = num1 + num2;

    alert("The sum of " + num1 + " and " + num2 + " is: " + sum);

}


// Example usage of the sum function

const number1 = 5;  // Change this value as needed

const number2 = 10; // Change this value as needed

calculateSum(number1, number2);
```

OUTPUT

Hello, World!

The sum of 5 and 10 is: 15

B.

```
// Create an array of 5 cities
let cities = ["New York", "Los Angeles", "Chicago", "Houston", "Phoenix"];

// Log the total number of cities
console.log("Total number of cities:", cities.length);

// Add a new city at the end
cities.push("San Francisco");
console.log("Cities after adding a new city:", cities);

// Remove the first city
cities.shift();
console.log("Cities after removing the first city:", cities);

// Find and log the index of a specific city
const specificCity = "Chicago"; // Change this to any city you want to search for
const index = cities.indexOf(specificCity);

if (index !== -1) {
    console.log("The index of " + specificCity + " is:", index);
} else {
    console.log(specificCity + " is not found in the array.");
```

}

OUTPUT

Total number of cities: 5

Cities after adding a new city: [

'New York',

'Los Angeles',

'Chicago',

'Houston',

'Phoenix',

'San Francisco'

]

Cities after removing the first city: ['Los Angeles', 'Chicago', 'Houston', 'Phoenix', 'San Francisco']

The index of Chicago is: 1

2. Read a string from the user, Find its length. Extract the word "JavaScript" using substring () or slice(). Replace one word with another word and log the new string. Write a function is Palindrome(str) that checks if a given string is a palindrome (reads the same backward).

```
// Function to check if a string is a palindrome

function isPalindrome(str) {

   const cleanedStr = str.replace(/[\W_]/g, '').toLowerCase(); // Remove non-alphanumeric characters and convert to lowercase

   const reversedStr = cleanedStr.split('').reverse().join('');

   return cleanedStr === reversedStr;

}


// Read a string from the user (for demonstration purposes, we'll use a prompt)

const userInput = prompt("Please enter a string:");


// Find the length of the string

const length = userInput.length;

console.log("Length of the string:", length);


// Extract the word "JavaScript" using substring() or slice()

// Assuming the string contains "JavaScript" for this example

const startIndex = userInput.indexOf("JavaScript");

const extractedWord = startIndex !== -1 ? userInput.substring(startIndex, startIndex + "JavaScript".length) : "Not found";

console.log("Extracted word:", extractedWord);


// Replace one word with another word

const wordToReplace = "word"; // Change this to the word you want to replace
```

```
const newWord = "phrase"; // Change this to the new word

const newString = userInput.replace(new RegExp(wordToReplace, 'g'), newWord);

console.log("New string after replacement:", newString);


// Check if the user input is a palindrome

const palindromeCheck = isPalindrome(userInput);

console.log(`Is the string a palindrome? ${palindromeCheck ? "Yes" : "No"}`);
```

OUTPUT

```
Please enter a string: FULLSTACK

Length of the string: 9

Extracted word: Not found

New string after replacement: FULLSTACK

Is the string a palindrome? No

Please enter a string:MOM

Length of the string: 3

Extracted word: Not found

New string after replacement: MOM

Is the string a palindrome? Yes
```

3. Create an object student with properties: name (string), grade (number), subjects (array), displayInfo() (method to log the student's details) Write a script to dynamically add a passed property to the student object, with a value of true or false based on their grade. Create a loop to log all keys and values of the student object.

```
// Create a student object
let student = {
    name: "John Doe",
    grade: 85,
    subjects: ["Math", "Science", "English"],

    // Method to display student's details
    displayInfo: function() {
        console.log("Student Name:", this.name);
        console.log("Grade:", this.grade);
        console.log("Subjects:", this.subjects.join(", "));
    }
};

// Function to dynamically add a property based on grade
function addGradeProperty(student) {
    // Set the property name based on grade criteria
    const propertyName = student.grade >= 60 ? 'passed' : 'failed';
    student[propertyName] = true;
}

// Call the method to display student info
```

```
student.displayInfo();


// Dynamically add the property
addGradeProperty(student);


// Log all keys and values of the student object
console.log("\nStudent Object Details:");
for (const key in student) {
    if (student.hasOwnProperty(key)) {
        console.log(key + ": " + student[key]);
    }
}
```

OUTPUT

```
Student Name: John Doe
Grade: 85
Subjects: Math, Science, English
Student Object Details:
name: John Doe
grade: 85
subjects: Math, Science, English
passed: true
```

4. Create a button in your HTML with the text "Click Me". Add an event listener to log "Button clicked!" to the console when the button is clicked. Select an image and add a mouseover event listener to change its border color. Add an event listener to the document that logs the key pressed by the user.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Event Listeners Example</title>
    <style>
        /* Styling for the image */
        #myImage {
            width: 300px;  /* Set your desired width */
            border: 2px solid transparent;  /* Default border */
            transition: border-color 0.3s;  /* Smooth transition for border color */
        }
    </style>
</head>
<body>

    <!-- Button with text "Click Me" -->
    <button id="clickMeButton">Click Me</button>

    <!-- Image element (please replace the src with a valid image URL) -->
    <img id="myImage" src="https://via.placeholder.com/300" alt="Placeholder Image">
```

```
    <script>
        // Add event listener to the button
        document.getElementById("clickMeButton").addEventListener("click", function() {
            console.log("Button clicked!");
        });

        // Add mouseover event listener to the image
        const image = document.getElementById("myImage");
        image.addEventListener("mouseover", function() {
            this.style.borderColor = "blue";  // Change border color on mouseover
        });
        image.addEventListener("mouseout", function() {
            this.style.borderColor = "transparent";  // Reset border color on mouseout
        });

        // Add keypress event listener to the document
        document.addEventListener("keydown", function(event) {
            console.log("Key pressed:", event.key);
        });
    </script>

</body>
</html>
```

5. Build a React application to track issues. Display a list of issues (use static data). Each issue should have a title, description, and status (e.g., Open/Closed). Render the list using a functional component.

Step 1: Set Up a React Application

If you haven't already set up a React application, you can use Create React App to do so. Open your terminal and run:

npx create-react-app issue-tracker

cd issue-tracker

Step 2: Create the Issue Tracker Component

1. Open the src folder in your React application.
2. Create a new file named IssueTracker.js.

Step 3: Add the Code

In the IssueTracker.js file, add the following code:

```
import React from 'react';


// Static data for issues
const issues = [
  {
    id: 1,
    title: 'Issue 1',
    description: 'Description for Issue 1',
    status: 'Open'
  },
  {
    id: 2,
    title: 'Issue 2',
    description: 'Description for Issue 2',
    status: 'Closed'
```

```
  },
  {
    id: 3,
    title: 'Issue 3',
    description: 'Description for Issue 3',
    status: 'Open'
  },
];

// Functional component to render the list of issues
const IssueTracker = () => {
  return (
    <div>
      <h1>Issue Tracker</h1>
      <ul>
        {issues.map(issue => (
          <li key={issue.id}>
            <h2>{issue.title}</h2>
            <p>{issue.description}</p>
            <p>Status: {issue.status}</p>
          </li>
        ))}
      </ul>
    </div>
  );
```

```
};


export default IssueTracker;
```

Step 4: Update the Main Application File

Now, open the src/App.js file and update it to include the IssueTracker component:

```
import React from 'react';
import './App.css';
import IssueTracker from './IssueTracker';


function App() {
  return (
    <div className="App">
      <IssueTracker />
    </div>
  );
}


export default App;
```


Step 5: Start the Application

Now you can start your React application. Open the terminal in your project directory and run:

```
npm start
```

6. Create a component Counter with A state variable count initialized to 0. Create Buttons to increment and decrement the count. Simulate fetching initial data for the Counter component using useEffect (functional component) or componentDidMount (class component). Extend the Counter component to Double the count value when a button is clicked. Reset the count to 0 using another button.

Step 1: Set Up a React Application

If you haven't already set up a React application, you can create one using Create React App:

npx create-react-app counter-app

cd counter-app

Step 2: Create the Counter Component

1. Open the src folder in your React application.
2. Create a new file named Counter.js.

Step 3: Add the Code

In the Counter.js file, add the following code:

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


const Counter = () => {
  // Initialize count state variable
  const [count, setCount] = useState(0);


  // Simulate fetching initial data
  useEffect(() => {
    const fetchData = async () => {
      // Simulating an API call
      const initialCount = await new Promise(resolve => setTimeout(() => resolve(0), 1000));
      setCount(initialCount);
```

```
    };

    fetchData();

  }, []);


  // Function to double the count
  const doubleCount = () => {
    setCount(count * 2);
  };


  return (
    <div>
      <h1>Counter: {count}</h1>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setCount(count - 1)}>Decrement</button>
      <button onClick={doubleCount}>Double</button>
      <button onClick={() => setCount(0)}>Reset</button>
    </div>
  );
};

export default Counter;
```

Step 4: Update the Main Application File

Now, open the src/App.js file and update it to include the Counter component:

```
import React from 'react';
import './App.css';
import Counter from './Counter';

function App() {
  return (
    <div className="App">
      <Counter />
    </div>
  );
}

export default App;
```


Step 5: Start the Application

You can now start your React application. Open the terminal in your project directory and run:

```
npm start
```

7. Install Express (npm install express). Set up a basic server that responds with "Hello, Express!" at the root endpoint (GET /). Create a REST API. Implement endpoints for a Product resource: GET : Returns a list of products. POST : Adds a new product. GET /:id: Returns details of a specific product. PUT /:id: Updates an existing product. DELETE /:id: Deletes a product. Add middleware to log requests to the console. Use express.json() to parse incoming JSON payloads.

Step 1: Set Up Your Project

1. Create a new directory for your project and navigate into it:

mkdir express-api

cd express-api

Initialize a new Node.js project:

npm init –y

Install Express:

npm install express

Step 2: Create the Server

1. Create a file named server.js in your project directory.
2. Add the following code to server.js:

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

// Middleware to log requests
app.use((req, res, next) => {
    console.log(`${req.method} ${req.url}`);
    next();
});
```

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

// Sample in-memory product data
let products = [
    { id: 1, name: 'Product 1', price: 100 },
    { id: 2, name: 'Product 2', price: 200 },
];

// Root endpoint
app.get('/', (req, res) => {
    res.send('Hello, Express!');
});

// REST API for Product resource

// GET: Returns a list of products
app.get('/products', (req, res) => {
    res.json(products);
});

// POST: Adds a new product
app.post('/products', (req, res) => {
    const newProduct = {
        id: products.length + 1, // Simple ID assignment
```

```
        name: req.body.name,
        price: req.body.price,
    };
    products.push(newProduct);
    res.status(201).json(newProduct);
});

// GET /:id: Returns details of a specific product
app.get('/products/:id', (req, res) => {
    const product = products.find(p => p.id === parseInt(req.params.id));
    if (!product) {
        return res.status(404).send('Product not found');
    }
    res.json(product);
});

// PUT /:id: Updates an existing product
app.put('/products/:id', (req, res) => {
    const product = products.find(p => p.id === parseInt(req.params.id));
    if (!product) {
        return res.status(404).send('Product not found');
    }
    product.name = req.body.name;
    product.price = req.body.price;
    res.json(product);
```

```
});


// DELETE /:id: Deletes a product
app.delete('/products/:id', (req, res) => {
    const productIndex = products.findIndex(p => p.id === parseInt(req.params.id));
    if (productIndex === -1) {
        return res.status(404).send('Product not found');
    }
    products.splice(productIndex, 1);
    res.status(204).send();
});


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

Step 3: Run Your Server

In your terminal, run the following command to start the server:

```
node server.js
```

Testing Your API

You can use tools like Postman or cURL to test the API endpoints:

1. **GET /**: Visit http://localhost:3000/ to see "Hello, Express!".
2. **GET /products**: Visit http://localhost:3000/products to see the list of products.
3. **POST /products**: Use Postman or cURL to send a POST request with a JSON body to http://localhost:3000/products:

{

"name": "Product 3",

"price": 300

}

1. **GET /products/:id**: Visit http://localhost:3000/products/1 (or any valid ID) to see the details of a specific product.

2. **PUT /products/:id**: Use Postman or cURL to send a PUT request to update a product:

{

"name": "Updated Product 1",

"price": 150

}

1. **DELETE /products/:id**: Send a DELETE request to http://localhost:3000/products/1 to delete a product.

8. Install the MongoDB driver for Node.js. Create a Node.js script to connect to the shop database. Implement insert, find, update, and delete operations using the Node.js MongoDB driver. Define a product schema using Mongoose. Insert data into the products collection using Mongoose. Create an Express API with a /products endpoint to fetch all products. Use fetch in React to call the /products endpoint and display the list of products. Add a POST /products endpoint in Express to insert a new product. Update the Product List, After adding a product, update the list of products displayed in React.

Step 1: Set Up Your Node.js Project

1. **Create a new directory for your project:**

mkdir shop-api

cd shop-api

Initialize a new Node.js project:

npm init –y

Install the required packages:

npm install express mongoose dotenv

Install the MongoDB driver:

npm install mongodb

Step 2: Set Up Environment Variables

1. **Create a .env file in the project root and add your MongoDB connection string:**

MONGODB_URI=mongodb://localhost:27017/shop

Step 3: Create the MongoDB Connection and Mongoose Schema

1. **Create a file named server.js:**

```
// server.js

const express = require('express');

const mongoose = require('mongoose');

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

```
dotenv.config();


const app = express();

const PORT = 3000;


// Middleware to parse JSON

app.use(express.json());


// Connect to MongoDB

mongoose.connect(process.env.MONGODB_URI, { useNewUrlParser: true, useUnifiedTopology: true })

  .then(() => console.log('MongoDB connected'))

  .catch(err => console.error('MongoDB connection error:', err));


// Define a product schema

const productSchema = new mongoose.Schema({

  name: { type: String, required: true },

  price: { type: Number, required: true }

});


// Create a Product model

const Product = mongoose.model('Product', productSchema);


// REST API Endpoints
```

```
// GET /products: Fetch all products
app.get('/products', async (req, res) => {
  try {
    const products = await Product.find();
    res.json(products);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

// POST /products: Insert a new product
app.post('/products', async (req, res) => {
  const product = new Product({
    name: req.body.name,
    price: req.body.price
  });

  try {
    const newProduct = await product.save();
    res.status(201).json(newProduct);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
});
```

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

Step 4: Run Your MongoDB Server

Ensure that your MongoDB server is running on localhost:27017. If you haven't installed MongoDB, you can download it from MongoDB's official website.

Step 5: Seed the Database (Optional)

You can optionally create a script to seed your database with initial products. Create a file named seed.js:

```
// seed.js
const mongoose = require('mongoose');
const dotenv = require('dotenv');

dotenv.config();

mongoose.connect(process.env.MONGODB_URI, { useNewUrlParser: true, useUnifiedTopology: true })
    .then(() => console.log('MongoDB connected'))
    .catch(err => console.error('MongoDB connection error:', err));

const productSchema = new mongoose.Schema({
    name: { type: String, required: true },
    price: { type: Number, required: true }
});

const Product = mongoose.model('Product', productSchema);
```

```
const seedProducts = async () => {
  await Product.deleteMany({});
  const products = [
    { name: 'Product 1', price: 100 },
    { name: 'Product 2', price: 200 },
    { name: 'Product 3', price: 300 },
  ];

  await Product.insertMany(products);
  console.log('Database seeded!');
  mongoose.connection.close();
};

seedProducts();
```

Run this script to seed your database:

```
node seed.js
```

Step 6: Create the React Application

1. **Create a new React application:**

```
npx create-react-app shop-client
cd shop-client
```

Install Axios for making HTTP requests:

```
npm install axios
```

1. **Update src/App.js to fetch and display products:**

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

const App = () => {
  const [products, setProducts] = useState([]);
  const [newProduct, setNewProduct] = useState({ name: '', price: '' });

  // Fetch products from the API
  const fetchProducts = async () => {
    const response = await axios.get('http://localhost:3000/products');
    setProducts(response.data);
  };

  useEffect(() => {
    fetchProducts();
  }, []);

  // Handle form submission to add a new product
  const handleSubmit = async (e) => {
    e.preventDefault();
    const response = await axios.post('http://localhost:3000/products', newProduct);
    setProducts([...products, response.data]);
```

```
    setNewProduct({ name: '', price: '' }); // Reset form
};

return (
    <div>
        <h1>Product List</h1>
        <ul>
            {products.map(product => (
                <li key={product._id}>{product.name} - ${product.price}</li>
            ))}
        </ul>

        <h2>Add a New Product</h2>
        <form onSubmit={handleSubmit}>
            <input
                type="text"
                placeholder="Product Name"
                value={newProduct.name}
                onChange={(e) => setNewProduct({ ...newProduct, name: e.target.value })}
                required
            />
            <input
                type="number"
                placeholder="Product Price"
                value={newProduct.price}
```

```
                onChange={(e) => setNewProduct({ ...newProduct, price: e.target.value })}
                required
            />
            <button type="submit">Add Product</button>
        </form>
    </div>
  );
};

export default App;
```

Step 7: Run the React Application

1. **Start the React application:**

```
npm start
```