Sarvajanik College of Engineering & Technology

Department of Information Technology

SUBJECT FILE

Subject Coordinator: Prof. Tushar Gohil

Name of Subject: Advanced Web Technology

Subject Code: BTIT14603

Year: BE-III

Semester: VI

Academic Year: 2023-24

Term Date: 26/12/2023 to 27/04/2024

Address
Dr. R. K. Desai Marg, Athwalines, Surat
www.scet.ac.in

Index

Teaching Scheme

- **Prerequisite:**

  Web Technologies

- **Rationale:**
  - Today's world is driven by Internet based applications.
  - The rationale behind this course is to impart the knowledge of java script-based framework for web programming among students of Information Technology.
  - Students will learn advanced web programming concepts related to Java script, React JS, Node JS, and MongoDB.

- **Teaching and Examination Scheme:**

Teaching Scheme				Theory Marks			Practical Marks		Total
L	**T**	**P**	**C**	**TEE**	**CA1**	**CA2**	**TEP**	**CA3**	
3	0	2	4	60	25	15	30	20	150

CA1: Continuous Assessment (assignments/projects/open book tests/closed book tests) CA2: Sincerity in attending classes/class tests/ timely submissions of assignments/self-learning attitude/solving advanced problems TEE: Term End Examination TEP: Term End Practical Exam (Performance and viva on practical skills learned in course) CA3: Regular submission of Lab work/Quality of work submitted/Active participation in lab sessions/viva on practical skills learned in course

Syllabus

Sr. No.	Contents	Total Hrs.	% Weightage
1.	**Refreshing JavaScript and CSS** CSS syntax, benefits, Responsive design, Bootstrap introduction, Java script syntax, Java script inbuilt objects, Error handling and event handling, DOM, Asynchronous Programming	08	17
2.	**React JS** Pure React, Props, State and the Component Tree, React Router, React Hooks, React Context, React Error Boundaries	12	28
3.	**Node JS** Introduction, File Access, Rest API, Events and Event Loop, timers, Error Handling, Networking	10	22
4.	**Express JS** Basics, Routing, Middleware, Template Engine	05	11
5.	**Mongo DB** Introduction to MongoDB,Mapping Relational database to MongoDB, MongoDB installation and configuration in Windows, MongoDB Create database, MongoDB Drop Database, MongoDB Create collection, MongoDB Drop collection ,MongoDB Insert Document, MongoDB Query Document, MongoDB Update Document,Delete document in MongoDB	05	11
6.	**Single Page Applications** Introduction to single page applications, designing and developing single page applications using MERN stack.	05	11

- Web link ((As per syllabus)
- Video Link (As per syllabus)
- Software (As per syllabus)

- Course Outcomes

Course Outcome	Statement	Weightage (%)
CO1	Explain the concepts of client-side programming using CSS and Java Script	10
CO2	Apply the concepts of React JS to extend basic HTML features	30
CO3	Utilize Node JS framework to build dynamic server-side applications	30
CO4	Build the Applications utilizing functionalities of Databases like Mongo DB.	20
CO5	Design and implement full featured single page application using MERN Stack.	10

Lecture Plan

Faculties:	Prof. Tushar Gohil (3 Hrs.) (TUE, WED, THU)

No.	Topics	Plan Date	Actual Date	Mode (B/M) *
Chapter 1: Refreshing Javascript and CSS (8 Hrs)				
1.	CSS syntax, benefits	26/12/23	02/01/24	B + M
2.	Responsive design, Bootstrap introduction	27/12/23	03/01/24	B + M
3.	Java script syntax	28/12/23	04/01/24	B + M
4.	Java script inbuilt objects	02/01/24	04/01/24	B + M
5.	Error handling and event handling	03/01/24	09/01/24	B + M
6.	DOM	04/01/24	09/01/24	B + M
7.	Asynchronous Programming	09/01/24	10/01/24	B + M
8.	Asynchronous Programming	10/01/24	11/01/24	B + M
Chapter 2: React JS (12 hrs)				
9.	Pure React,	11/01/24	16/01/24	B + M
10.	Props	16/01/24	17/01/24	B + M
11.	Props	17/01/24	18/01/24	B + M
12.	State and the Component Tree	18/01/24	24/01/24	B + M
13.	State and the Component Tree	23/01/24	25/01/24	B + M
14.	React Router	24/01/24	30/01/24	B + M
15.	React Hooks	25/01/24	01/02/24	B + M
16.	React Hooks	01/02/24	06/02/24	B + M
17.	React Hooks	06/02/24	07/02/24	B + M
18.	React Context	07/02/24	07/02/24	B + M
19.	React Context	08/02/24	08/02/24	B + M
20.	React Error Boundaries	13/02/24	13/02/24	B + M

Chapter 3: Node JS (10 hrs)				
21.	Introduction	14/02/24	14/02/24	B + M
22.	File Access	15/02/24	15/02/24	B + M
23.	Rest API	20/02/24	20/02/24	B + M
24.	Rest API	21/02/24	21/02/24	B + M
25.	Events and Event Loop	22/02/24	22/02/24	B + M
26.	Events and Event Loop	27/02/24	27/02/24	B + M
27.	Timers	28/02/24	28/02/24	B + M
28.	Error Handling	29/02/24	29/02/24	B + M
29.	Networking	05/03/24	05/03/24	B + M
30.	Networking	06/03/24	06/03/24	B + M
Chapter 4: Express JS (5 Hrs)				
31.	Basics	07/03/24	07/03/24	B + M
32.	Routing	12/03/24	02/04/24	B + M
33.	Routing	13/03/24	04/04/24	B + M
34.	Middleware	14/03/24	09/04/24	B + M
35.	Template Engine	27/03/24	10/04/24	B + M
Chapter 5: MongoDB (5 Hrs)				
36.	Introduction to MongoDB, Mapping Relational database to MongoDB	28/03/24	11/04/24	B + M
37.	MongoDB installation and configuration in Windows	02/04/24	11/04/24	B + M
38.	MongoDB Create database, MongoDB Drop Database, MongoDB Create collection	03/04/24	16/04/24	B + M
39.	MongoDB Drop collection ,MongoDB Insert Document, MongoDB Query Document	04/04/24	16/04/24	B + M
40.	MongoDB Update Document, Delete document in MongoDB	09/04/24	16/04/24	B + M
Chapter 6: Single Page Applications (5 Hrs)				
41.	Introduction to single page applications	10/04/24	17/04/24	B + M
42.	Designing and developing single page applications using MERN stack.	11/04/24	17/04/24	B + M
43.	Designing and developing single page applications using MERN stack.	16/04/24	18/04/24	B + M

44.	Designing and developing single page applications using MERN stack.	17/04/24	18/04/24	B + M
45.	Designing and developing single page applications using MERN stack.	18/04/24	18/04/24	B + M

* B - Black Board, M - Multi-media

Lab Plan

Faculty:	Prof. Tushar Gohil (4 Hrs)	Batch:	A, B
	Prof. Palak Desai (2 Hrs.)		A
	Prof. Forum Patel (2 Hrs.)		B

Practical List

Sr No	Problem Statement	Batch – A		Batch-B	
		Planned Date	Actual Date	Planned Date	Actual Date
01	Create a simple guess for the number type game. It should choose a random number between 1 and 100, then challenge the player to guess the number in 10 turns. After each turn the player should be told if they are right or wrong, and if they are wrong, whether the guess was too low or too high. It should also tell the player what numbers they previously guessed. The game will end once the player guesses correctly, or once they run out of turns. When the game ends, the player should be given an option to start playing again.	01/01/24	01/01/24	01/01/24	01/01/24
02	Create a Hello World Web Page using React JS.	08/01/24	08/01/24	08/01/24	08/01/24
03	Create a Sign-Up Form in React JS. The Sign-up form should ask for Name, Mobile, Email, Address, Date of Birth, Gender, Username, Password and Confirm Password. The Form should have two buttons, one for reset and one for submitting. The Submit Button should be enabled only when all the input values are validated.	22/01/24	22/01/24	22/01/24	22/01/24
04	Develop a simple application using React.js where a user can Add/Delete notes. Each note timestamped as well.	29/01/24	29/01/24	29/01/24	29/01/24
05	Create a to-do list application using react-context.	05/02/24	05/02/24	05/02/24	05/02/24
06	Create a Hello World Nodejs Application.	12/02/24	12/02/24	12/02/24	12/02/24
07	Create an Http Server which will respond with message "Welcome to the World of Nodejs" to the client.	26/02/24	26/02/24	26/02/24	26/02/24
08	Create an Http Server which will respond with current date and time to the client	04/03/24	04/03/24	04/03/24	04/03/24
09	Demonstrate the working of cookies and sessions in nodejs.	11/03/24	11/03/24	11/03/24	11/03/24
10	Create a Signup Web Applications using React JS,nodejs and express framework. The Sign-up form should ask for Name, Mobile, Email, Address, Date of Birth, Gender, Username, Password and Confirm Password. The Form should have two buttons, one for reset and one for submitting. The Submit Button should be enabled only when all the input values are validated. Upon Clicking the Submit Button Your webpage should traversed to a route "/process_request" which is defined using NodeJS and	01/04/24	01/04/24	01/04/24	01/04/24

	express framework and which will display the contents back to the client and insert all the data to the database made in mongo dB.				
11	Create a Single Page Application using MERN Stack.	08/04/24	08/04/24	08/04/24	08/04/24

Rubrics

Category	**Level of Performance**		
	10-8 marks	7-4 marks	3-1 marks
Implementation (10)	Able to implement in time limit with correct result	Able to implement in time limit, no proper result	Able to implement with error and no result
	5-4 marks	3-2 marks	1 marks
Viva (5)	Able to interpret practical theoretically as well as practically	Partially able to interpret theoretically as well as practically	Unable to interpret theoretical as well practical concepts
Documentation Level			
	5-4 marks	3-2 marks	1 marks
File Submission (5)	Completed on time	Incomplete but on time	Incomplete and delay

Reference Books

1. Node.js in Action, Alex Young, Bradley Meck, Mike Cantelon, Tim Oxley, Marc Harter, T.J. Holowaychuk, Nathan Rajlich
2. Node.Js in Practice, Alex Young, Marc Harter, Ben Noordhuis
3. Professional Node.js, Pedro Teixeira
4. The Road to Learn React: Your Journey to Master Plain Yet Pragmatic React.Js, Robin Wieruch
5. MongoDB Cookbook, Cyrus Dasadia , Amol Nayak

1. Guess the Number Game in JavaScript

Objective:

- To create a web-based game where the player guesses a randomly chosen number between 1 and 100 within 10 turns.

Materials:

- Text editor or IDE
- Web browser

Procedure:

1. **Create Project Structure:**
   - Create a new directory for the project.
   - Inside this directory, create an `index.html` file and a `script.js` file.
2. **HTML Setup (`index.html`):**

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Guess the Number Game</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin-top: 50px;
        }
    </style>
</head>
<body>
    <h1>Guess the Number</h1>
    <p>Guess a number between 1 and 100:</p>
    <input type="number" id="guessInput" min="1" max="100">
    <button id="guessButton">Guess</button>
    <p id="message"></p>
    <p id="previousGuesses"></p>
    <button id="restartButton" style="display:none;">Restart</button>
    <script src="script.js"></script>
</body>
</html>
```

3. **JavaScript Setup (`script.js`):**

```
javascript
Copy code
let randomNumber = Math.floor(Math.random() * 100) + 1;
let guesses = [];
let maxTurns = 10;
```

```
document.getElementById('guessButton').addEventListener('click', () => {
    const guessInput = document.getElementById('guessInput');
    const guess = Number(guessInput.value);
    if (!guess || guess < 1 || guess > 100) {
        alert('Please enter a valid number between 1 and 100.');
        return;
    }
    guesses.push(guess);

    let message = '';
    if (guess < randomNumber) {
        message = 'Too low!';
    } else if (guess > randomNumber) {
        message = 'Too high!';
    } else {
        message = 'Congratulations! You guessed it right!';
        endGame();
        return;
    }

    if (guesses.length >= maxTurns) {
        message = `Game over! The correct number was ${randomNumber}.`;
        endGame();
        return;
    }

    document.getElementById('message').textContent = message;
    document.getElementById('previousGuesses').textContent = `Previous
guesses: ${guesses.join(', ')}`;
    guessInput.value = '';
    guessInput.focus();
});

document.getElementById('restartButton').addEventListener('click', () => {
    randomNumber = Math.floor(Math.random() * 100) + 1;
    guesses = [];
    document.getElementById('message').textContent = '';
    document.getElementById('previousGuesses').textContent = '';
    document.getElementById('restartButton').style.display = 'none';
    document.getElementById('guessInput').disabled = false;
    document.getElementById('guessButton').disabled = false;
});

function endGame() {
    document.getElementById('guessInput').disabled = true;
    document.getElementById('guessButton').disabled = true;
    document.getElementById('restartButton').style.display = 'block';
}
```

Expected Outcomes:

- The player will be able to guess the number within 10 turns.
- The player will receive feedback after each guess indicating whether the guess was too low, too high, or correct.
- The player will see their previous guesses.
- The game will end when the player guesses correctly or runs out of turns, with an option to restart the game.

2. Hello World Web Page using React JS

Objective:

- To create a simple web page that displays "Hello World" using React JS.

Materials:

- Node.js installed
- Text editor or IDE
- Web browser

Procedure:

1. **Setup Project Structure:**
   - Open your terminal and create a new directory for the project. Navigate into this directory.
   - Use Create React App to set up a new React project:

     ```
     npx create-react-app hello-world-react
     cd hello-world-react
     ```

2. **Modify the App Component:**
   - Open the project in your text editor or IDE.
   - Navigate to `src/App.js` and replace its content with the following code:

     ```
     import React from 'react';

     function App() {
       return (
         <div>
           <h1>Hello World</h1>
         </div>
       );
     }

     export default App;
     ```

3. **Run the React Application:**
   - In your terminal, start the development server:

     ```
     npm start
     ```

   - This will open a new browser window/tab with the default URL `http://localhost:3000`, and you should see the "Hello World" message displayed.

Expected Outcomes:

- The web page will display "Hello World" in the browser.

Additional Notes:

- The development server provided by Create React App supports hot reloading, so any changes you make to the code will be reflected immediately in the browser without needing to refresh the page.

- The structure of a Create React App project includes several files and folders, but for this basic "Hello World" example, the main file you are concerned with is `src/App.js`.

3. Creating a Sign-Up Form in React JS

Objective:

- To create a sign-up form in React JS that includes fields for Name, Mobile, Email, Address, Date of Birth, Gender, Username, Password, and Confirm Password. The form will have reset and submit buttons, with the submit button enabled only when all inputs are validated.

Materials:

- Node.js installed
- Text editor or IDE
- Web browser

Procedure:

1. **Setup Project Structure:**
   - Open your terminal and create a new React project using Create React App:

```
npx create-react-app signup-form
cd signup-form
```

2. **Create the Sign-Up Form Component:**
   - In your `src` directory, create a new file `SignupForm.js` and add the following code:

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

const SignupForm = () => {
  const [formData, setFormData] = useState({
    name: '',
    mobile: '',
    email: '',
    address: '',
    dateOfBirth: '',
    gender: '',
    username: '',
    password: '',
    confirmPassword: '',
  });

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

  const validate = () => {
    const newErrors = {};

    if (!formData.name) newErrors.name = 'Name is required';
    if (!formData.mobile) newErrors.mobile = 'Mobile number is
required';
    if (!formData.email) newErrors.email = 'Email is required';
    else if (!/\S+@\S+\.\S+/.test(formData.email)) newErrors.email =
'Email is invalid';
    if (!formData.address) newErrors.address = 'Address is required';
    if (!formData.dateOfBirth) newErrors.dateOfBirth = 'Date of Birth
is required';
    if (!formData.gender) newErrors.gender = 'Gender is required';
    if (!formData.username) newErrors.username = 'Username is
required';
    if (!formData.password) newErrors.password = 'Password is
required';
```

```
    if (formData.password !== formData.confirmPassword)
newErrors.confirmPassword = 'Passwords do not match';

    setErrors(newErrors);
    return Object.keys(newErrors).length === 0;
  };

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({
      ...formData,
      [name]: value,
    });
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    if (validate()) {
      console.log('Form data:', formData);
      alert('Form submitted successfully!');
      setFormData({
        name: '',
        mobile: '',
        email: '',
        address: '',
        dateOfBirth: '',
        gender: '',
        username: '',
        password: '',
        confirmPassword: '',
      });
    }
  };

  const handleReset = () => {
    setFormData({
      name: '',
      mobile: '',
      email: '',
      address: '',
      dateOfBirth: '',
      gender: '',
      username: '',
      password: '',
      confirmPassword: '',
    });
    setErrors({});
  };

  return (
    <form onSubmit={handleSubmit}>
      <div>
        <label htmlFor="name">Name</label>
        <input type="text" name="name" value={formData.name}
onChange={handleChange} />
        {errors.name && <div>{errors.name}</div>}
      </div>

      <div>
        <label htmlFor="mobile">Mobile</label>
        <input type="text" name="mobile" value={formData.mobile}
onChange={handleChange} />
        {errors.mobile && <div>{errors.mobile}</div>}
      </div>
```

```
        <div>
          <label htmlFor="email">Email</label>
          <input type="text" name="email" value={formData.email}
onChange={handleChange} />
          {errors.email && <div>{errors.email}</div>}
        </div>

        <div>
          <label htmlFor="address">Address</label>
          <input type="text" name="address" value={formData.address}
onChange={handleChange} />
          {errors.address && <div>{errors.address}</div>}
        </div>

        <div>
          <label htmlFor="dateOfBirth">Date of Birth</label>
          <input type="date" name="dateOfBirth"
value={formData.dateOfBirth} onChange={handleChange} />
          {errors.dateOfBirth && <div>{errors.dateOfBirth}</div>}
        </div>

        <div>
          <label htmlFor="gender">Gender</label>
          <select name="gender" value={formData.gender}
onChange={handleChange}>
            <option value="">Select</option>
            <option value="male">Male</option>
            <option value="female">Female</option>
            <option value="other">Other</option>
          </select>
          {errors.gender && <div>{errors.gender}</div>}
        </div>

        <div>
          <label htmlFor="username">Username</label>
          <input type="text" name="username" value={formData.username}
onChange={handleChange} />
          {errors.username && <div>{errors.username}</div>}
        </div>

        <div>
          <label htmlFor="password">Password</label>
          <input type="password" name="password"
value={formData.password} onChange={handleChange} />
          {errors.password && <div>{errors.password}</div>}
        </div>

        <div>
          <label htmlFor="confirmPassword">Confirm Password</label>
          <input type="password" name="confirmPassword"
value={formData.confirmPassword} onChange={handleChange} />
          {errors.confirmPassword && <div>{errors.confirmPassword}</div>}
        </div>

        <div>
          <button type="submit" disabled={!validate()}>Submit</button>
          <button type="button" onClick={handleReset}>Reset</button>
        </div>
      </form>
    );
};

export default SignupForm;
```

3. **Integrate the Sign-Up Form into Your App:**
   - Open `src/App.js` and modify it to include the `SignupForm` component:

```
import React from 'react';
import SignupForm from './SignupForm';

function App() {
  return (
    <div>
      <h1>Sign Up</h1>
      <SignupForm />
    </div>
  );
}

export default App;
```

4. **Run the React Application:**
   - In your terminal, start the development server:

```
npm start
```

   - This will open a new browser window/tab with the default URL `http://localhost:3000`, and you should see the sign-up form displayed.

Expected Outcomes:

- The sign-up form will display input fields for Name, Mobile, Email, Address, Date of Birth, Gender, Username, Password, and Confirm Password.
- The form will validate each input field and display error messages for invalid inputs.
- The submit button will be enabled only when all inputs are validated.
- The reset button will clear all input fields and error messages.

4. Creating a Notes Application Using React.js

Objective:

- To develop a simple React.js application where users can add and delete notes. Each note should be timestamped.

Materials:

- Node.js installed
- Text editor or IDE
- Web browser

Procedure:

1. **Setup Project Structure:**
   - Open your terminal and create a new React project using Create React App:

```
npx create-react-app notes-app
cd notes-app
```

2. **Create the Notes Component:**
   - In your `src` directory, create a new file `Notes.js` and add the following code:

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

const Notes = () => {
  const [notes, setNotes] = useState([]);
  const [noteText, setNoteText] = useState('');

  const handleAddNote = () => {
    if (noteText.trim()) {
      const newNote = {
        id: Date.now(),
        text: noteText,
        timestamp: new Date().toLocaleString()
      };
      setNotes([...notes, newNote]);
      setNoteText('');
    }
  };

  const handleDeleteNote = (id) => {
    setNotes(notes.filter(note => note.id !== id));
  };

  return (
    <div>
      <h1>Notes</h1>
      <div>
        <input
          type="text"
          value={noteText}
          onChange={(e) => setNoteText(e.target.value)}
          placeholder="Write a note..."
        />
        <button onClick={handleAddNote}>Add Note</button>
      </div>
      <ul>
```

```
          {notes.map(note => (
            <li key={note.id}>
              <div>
                <span>{note.text}</span>
                <small>{note.timestamp}</small>
              </div>
              <button onClick={() =>
handleDeleteNote(note.id)}>Delete</button>
            </li>
          ))}
        </ul>
      </div>
    );
};

export default Notes;
```

3. **Integrate the Notes Component into Your App:**
   - Open `src/App.js` and modify it to include the `Notes` component:

```
import React from 'react';
import Notes from './Notes';

function App() {
  return (
    <div>
      <Notes />
    </div>
  );
}

export default App;
```

4. **Add Some Basic Styling:**
   - Open `src/App.css` and add some basic styling for the notes application:

```
body {
  font-family: Arial, sans-serif;
  background-color: #f7f7f7;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  margin: 0;
}

div {
  background-color: #fff;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

input {
  width: calc(100% - 90px);
  padding: 10px;
  margin-right: 10px;
  border: 1px solid #ccc;
  border-radius: 5px;
}

button {
  padding: 10px 20px;
```

```
  border: none;
  background-color: #28a745;
  color: #fff;
  border-radius: 5px;
  cursor: pointer;
}

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

ul {
  list-style-type: none;
  padding: 0;
}

li {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 10px;
  border-bottom: 1px solid #ccc;
}

small {
  color: #999;
}

li button {
  background-color: #dc3545;
}

li button:hover {
  background-color: #c82333;
}
```

5. **Run the React Application:**
   - In your terminal, start the development server:

     ```
     npm start
     ```

   - This will open a new browser window/tab with the default URL `http://localhost:3000`, and you should see the notes application.

Expected Outcomes:

- The application will allow users to add notes with a timestamp.
- Users can delete notes by clicking the "Delete" button next to each note.
- The notes list will be displayed in a clean and user-friendly interface.

5. Creating a To-Do List Application Using React Context

Follow same procedure as mentioned in previous program.

6. Creating a "Hello World" Node.js Application

Objective:

- To create a basic Node.js application that responds with "Hello World".

Materials:

- Node.js installed
- Text editor or IDE
- Terminal or command prompt

1. **Setup Project Structure:**
   - Create a new directory for your project and navigate into it:

     ```
     mkdir hello-world-nodejs
     cd hello-world-nodejs
     ```

2. **Create the JavaScript File:**
   - Create a new file named `index.js` and add the following code:

     ```
     console.log('Hello World');
     ```

3. **Run the Script:**
   - In your terminal, run the Node.js script using the `node` command:

     ```
     node index.js
     ```

   - You should see the output `Hello World` printed in your terminal.

Expected Outcomes:

- Running the `index.js` file using Node.js will print `Hello World` to the console.

Troubleshooting:

- If you encounter any errors, ensure that Node.js is properly installed on your system.
- Verify that you are in the correct directory (`hello-world-nodejs`) when running `node index.js`.

Additional Notes:

- This example demonstrates a basic use of Node.js to run a simple script without setting up an HTTP server or handling HTTP requests.

7. Creating an HTTP Server in Node.js

Objective:

- To create a basic HTTP server in Node.js that responds with the message "Welcome to the World of Node.js" to clients requesting a specific route.

Materials:

- Node.js installed
- Text editor or IDE
- Terminal or command prompt

Procedure:

1. **Setup Project Structure:**
   - Create a new directory for your project and navigate into it:

     ```
     mkdir http-server-nodejs
     cd http-server-nodejs
     ```

2. **Initialize a New Node.js Project:**
   - Run the following command to initialize a new Node.js project and create a `package.json` file:

     ```
     npm init -y
     ```

3. **Create the Server File:**
   - Create a new file named `server.js` in the `http-server-nodejs` directory and add the following code:

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

     const hostname = '127.0.0.1';
     const port = 3000;

     const server = http.createServer((req, res) => {
       // Set HTTP status and headers
       res.statusCode = 200;
       res.setHeader('Content-Type', 'text/plain');

       // Send the response body
       res.end('Welcome to the World of Node.js\n');
     });

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

4. **Run the Server:**
   - In your terminal, start the Node.js server by executing the following command in the `http-server-nodejs` directory:

     ```
     node server.js
     ```

- You should see a message in your terminal indicating that the server is running:

  ```
  Server running at http://127.0.0.1:3000/
  ```

5. **Access the Server:**
   - Open your web browser or use a tool like `curl` and navigate to `http://127.0.0.1:3000/`.
   - You should see the response "Welcome to the World of Node.js" displayed in your browser or terminal.

Expected Outcomes:

- The Node.js application should start an HTTP server that listens on port 3000.
- When accessing `http://127.0.0.1:3000/` in a web browser or using `curl`, the message "Welcome to the World of Node.js" should be displayed.

8. Creating an HTTP Server in Node.js to Respond with Current Date and Time

Follow same procedure as mentioned in previous program.

9. Demonstrating Cookies and Sessions in Node.js

Objective:

- To demonstrate the usage of cookies and sessions in a Node.js application using the `express` framework.

Materials:

- Node.js installed
- Text editor or IDE
- Terminal or command prompt

Procedure:

1. **Setup Project Structure:**
   - Create a new directory for your project and navigate into it:

     ```
     mkdir cookies-sessions-demo
     cd cookies-sessions-demo
     ```

2. **Initialize a New Node.js Project:**
   - Run the following command to initialize a new Node.js project and create a `package.json` file:

     ```
     npm init -y
     ```

3. **Install Required Packages:**
   - Install `express` and `express-session` packages:

     ```
     npm install express express-session
     ```

4. **Create the Server File:**
   - Create a new file named `app.js` in the `cookies-sessions-demo` directory and add the following code:

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

     const app = express();
     const port = 3000;

     // Middleware for session management
     app.use(session({
       secret: 'your-secret-key',
       resave: false,
       saveUninitialized: true,
       cookie: { secure: false } // Change to true in production with HTTPS
     }));

     // Route for setting a session variable
     ```

```
app.get('/setSession', (req, res) => {
  req.session.username = 'JohnDoe';
  res.send('Session variable set');
});

// Route for getting the session variable
app.get('/getSession', (req, res) => {
  const username = req.session.username || 'No session variable set';
  res.send(`Session username: ${username}`);
});

// Route for clearing the session variable
app.get('/clearSession', (req, res) => {
  req.session.destroy((err) => {
    if (err) {
      console.error('Error destroying session:', err);
      res.send('Error clearing session');
    } else {
      res.send('Session cleared');
    }
  });
});

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

5. **Run the Server:**
   - In your terminal, start the Node.js server by executing the following command in the `cookies-sessions-demo` directory:

     ```
     node app.js
     ```

   - You should see a message in your terminal indicating that the server is running:

     ```
     Server running at http://localhost:3000
     ```

6. **Test the Routes:**
   - Open your web browser or use a tool like `curl` to test the following routes:
     - **Set Session Variable:**
       - Navigate to `http://localhost:3000/setSession`. This will set a session variable `username` with the value `'JohnDoe'`.
     - **Get Session Variable:**
       - Navigate to `http://localhost:3000/getSession`. This will retrieve and display the session variable `username`.
     - **Clear Session:**
       - Navigate to `http://localhost:3000/clearSession`. This will clear the session and destroy the session variable `username`.

Expected Outcomes:

- When navigating to `http://localhost:3000/setSession`, you should see the message "Session variable set".
- When navigating to `http://localhost:3000/getSession`, you should see the message "Session username: JohnDoe" after setting the session.
- When navigating to `http://localhost:3000/clearSession`, you should see the message "Session cleared".

10. Creating a Signup Web Application using React JS, Node.js, and Express

Objective:

- To develop a Signup Web Application that collects user information and submits it to a Node.js backend for processing and storage in a MongoDB database.

Materials:

- Node.js installed
- MongoDB installed and running
- Text editor or IDE
- Terminal or command prompt

Procedure:

1. **Setup Project Structure:**
   - Create a new directory for your project and navigate into it:

     ```
     mkdir signup-web-app
     cd signup-web-app
     ```

2. **Initialize Frontend (React JS):**
   - Create the frontend using Create React App or your preferred React setup:

     ```
     npx create-react-app client
     ```

   - Navigate into the `client` directory:

     ```
     cd client
     ```

3. **Create Signup Form Component:**
   - Within the React application (`client/src`), create a signup form component (`SignupForm.js`) that includes fields for Name, Mobile, Email, Address, Date of Birth, Gender, Username, Password, and Confirm Password.
   - Implement validation for the form fields (e.g., required fields, email format, password matching).
   - Include buttons for reset and submit. Ensure the submit button is enabled only when all fields are validated.
4. **Setup Backend (Node.js with Express):**
   - Navigate back to the root directory (`signup-web-app`):

     ```
     cd ..
     ```

   - Initialize the backend:

     ```
     npm init -y
     ```

   - Install necessary dependencies:

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

   - Create a `server.js` file for the Express server and define routes:

        - Implement routes for handling signup form submission (`POST /process_request`) which will validate inputs, store data in MongoDB, and send back a response to the client.
5. **Connect to MongoDB:**
    - Use Mongoose to connect your Node.js application to MongoDB. Ensure you have MongoDB installed and running locally or use a cloud-hosted MongoDB service.
    - Define a Mongoose schema for storing user information (Name, Mobile, Email, etc.).
6. **Implement Form Submission Handling:**
    - In the backend (`server.js`), handle form submissions from the React frontend.
    - Validate incoming data, hash passwords securely, and store validated user information in MongoDB.
    - Send a response back to the React frontend indicating success or failure of the signup process.
7. **Integrate Frontend with Backend:**
    - Update the React frontend (`client/src/SignupForm.js`) to send form data to the Node.js backend (`POST /process_request`) when the submit button is clicked.
    - Handle responses from the backend and provide feedback to the user.

11. Creating a Single Page Application (SPA) using MERN Stack

Objective:

- To develop a Single Page Application (SPA) using MongoDB, Express.js, React.js, and Node.js.

Materials:

- Node.js installed
- MongoDB installed and running
- Text editor or IDE
- Terminal or command prompt

Procedure:

1. **Setup Project Structure:**
   - Create a new directory for your project:

     ```
     mkdir mern-spa
     cd mern-spa
     ```

2. **Initialize Backend (Node.js with Express):**
   - Initialize a new Node.js project and navigate into it:

     ```
     npm init -y
     ```

   - Install necessary dependencies (e.g., express, mongoose, body-parser):

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

   - Create a `server.js` file for the Express server and define routes to serve the SPA and handle API requests.
3. **Connect to MongoDB:**
   - Use Mongoose to connect your Node.js application to MongoDB. Ensure you have MongoDB installed and running locally or use a cloud-hosted MongoDB service.
   - Define Mongoose schemas and models for storing data related to your application (e.g., users, posts).
4. **Initialize Frontend (React.js):**
   - Initialize a new React.js application within the project directory (e.g., `client` folder):

     ```
     npx create-react-app client
     ```

   - Navigate into the `client` directory:

     ```
     cd client
     ```

5. **Develop the SPA with React.js:**
   - Build out React components that will form the pages and components of your SPA.
   - Utilize React Router for navigation within the SPA, defining routes and handling page transitions.
   - Implement state management using React Context API or Redux to manage global application state.
6. **Integrate Frontend with Backend:**
   - Implement API calls from React components to the Node.js backend using `fetch`, Axios, or

other HTTP client libraries.
    - Define API endpoints on the backend (`server.js`) to handle CRUD operations (Create, Read, Update, Delete) for your application data.
7. **Implement Authentication (Optional):**
    - If your application requires user authentication, implement signup, login, and logout functionality.
    - Use JWT (JSON Web Tokens) for authentication and authorization purposes.
8. **Testing and Deployment:**
    - Test the SPA locally by running both the React frontend and Node.js backend simultaneously.
    - Deploy the SPA to a cloud platform (e.g., Heroku, AWS) for public access and scalability.
    - Implement logging and error handling to monitor and troubleshoot issues in production.

Expected Outcomes:

- You should have a fully functional Single Page Application (SPA) that integrates MongoDB for data storage, Express.js for API handling, React.js for frontend views, and Node.js for backend logic.
- The SPA should provide a seamless user experience with dynamic content updates and navigation within a single web page.

Mid-semester Question Paper

B.Tech –III (6th Semester) CA-1 (MIDTERM) EXAMINATION March 2024
Subject: Advanced Web Technology

Time: 09:30 AM TO 10.45 AM **Subject Code: BTIT13604**
Total Marks: 25 **Date: 21/03/2024**

Q:1	Describe the useState and useEffect hooks in React and their usage patterns with examples.	**4**
Q:2	**Attempt Any Four.** 1. Compare and contrast CSS and inline stylesheets, highlighting their advantages and disadvantages. 2. Illustrate the working of async/await in JavaScript. 3. Explain the concept of virtual DOM in React and its advantages. 4. Explain the significance of events and the event loop in Node.js applications. 5. Demonstrate how React Context can be used as an alternative to prop drilling for state management in deeply nested components.	**12**
Q:3	**Attempt Any Three.** 1. Write a React component that fetches data from an API using the useEffect hook and displays it in a list. 2. Create a React form component that captures user input for a username and password, and updates state accordingly. 3. Implement a simple TCP server using Node.js that listens on port 3000 and logs "Client connected" when a client connects. 4. Create a React component that uses a state to display a counter that increments by 1 every time a button is clicked. 5. Develop a Node.js script that recursively reads all files in a directory and its subdirectories and logs their names to the console.	**9**

B.Tech –III (6th Semester) CA-1 (MIDTERM) EXAMINATION March 2024
Subject: Advanced Web Technology

Time: 04:00 PM TO 05.15 PM **Subject Code: BTIT13604**
Total Marks: 25 **Date: 08/04/2024**

Q:1	Discuss the different types of React Hooks and their use cases.	**4**
Q:2	**Attempt Any Four.** 6. What is Chaining in Promise? Provide an Example. 7. Compare and Contrast Class Components and functional components in React. 8. Explain the concept of virtual DOM in React and its advantages. 9. Explain the significance of events and the event loop in Node.js applications. 10. Discuss the difference between props and state in React Components.	**12**
Q:3	**Attempt Any Three.** 6. Write a React component that fetches data from an API using the useEffect hook and displays it in a list. 7. Create a React form component that captures user input for a username and password, and updates state accordingly. 8. Implement a Node.js script to read the contents of a text file and display them in the console. 9. Create a basic REST API using Node.js that responds with "Hello, API!" when accessed via an HTTP GET request. 10. Write a Node.js script that uses setTimeout to print "Delayed Message" after 2 seconds.	**9**

****

Continuous Assessment Record

Printed Mark sheet (in the given format)

Result Analysis

Number of Students	Mid-I (21/03/24)
Total	75
Not allowed	00
Absent	01
Fail	00
Pass	74
Pass (%)	100 %
AA	49
AB	13
BB	05
BC	05
CC	03
CD	00
FF	00

***Note**: AA (>=85%), AB (>=75% < 85%),BB (>=65% < 75%), BC (>=55% < 65%), CC (>=45% < 55), CD (>=40% < 45%), FF (<45%)*

Counselling Report

Date	Enrollment No.	Name Of Students	Details

Attendance Record

Musters

Students Practical Files/Tutorials/Assignments/Other Records

Note:

1) Max 5 best records for student Files/Tutorial/Assignments
2) Other records – Term end practical exam supplementary / Quiz Hard copy or any other records (all the records need to be submitted)