1.Create a stopwatch app using the following ES6+ features: Use Arrow functions to start, stop, and reset the stopwatch. Use classes to define a Stopwatch with methods for starting, stopping, and resetting the timer.

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title> Lab Program 1</title>
</head>
<body>

   <div id="time" class="time">00:00:000</div>
   <button id="startBtn" class="start">Start</button>
   <button id="stopBtn" class="stop">Stop</button>
   <button id="resetBtn" class="reset">Reset</button>

  <script type="module">
   class Stopwatch {
    constructor(displayElement) {
     this.displayElement = displayElement;
     this.timer = null;
     this.startTime = 0;
     this.elapsedTime = 0;
    }
    formatTime = (ms) => {
     const minutes = String(Math.floor(ms / 60000)).padStart(2, "0");
     const seconds = String(Math.floor((ms % 60000) / 1000)).padStart(2, "0");
     const milliseconds = String(ms % 1000).padStart(3, "0");
     return `${minutes}:${seconds}:${milliseconds}`;
    };
    start = () => {
     if (this.timer) return; // avoid multiple intervals
     this.startTime = Date.now() - this.elapsedTime;
     this.timer = setInterval(() => {
      this.elapsedTime = Date.now() - this.startTime;
      this.displayElement.textContent = this.formatTime(this.elapsedTime);
     }, 10); // update every 10 ms
    };
    stop = () => {
     clearInterval(this.timer);
     this.timer = null;
    };
    reset = () => {
     this.stop();
     this.elapsedTime = 0;
     this.displayElement.textContent = "00:00:000";
```

```
    };
  }
  // DOM elements
  const timeDisplay = document.getElementById("time");
  const startBtn = document.getElementById("startBtn");
  const stopBtn = document.getElementById("stopBtn");
  const resetBtn = document.getElementById("resetBtn");

  // Stopwatch instance
  const stopwatch = new Stopwatch(timeDisplay);

  // Event listeners (arrow functions)
  startBtn.addEventListener("click", () => stopwatch.start());
  stopBtn.addEventListener("click", () => stopwatch.stop());
  resetBtn.addEventListener("click", () => stopwatch.reset());
 </script>
</body>
</html>
```

2. Build a simple calculator using ES6 features

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Simple ES6 Calculator</title>

  <style>

    body {

      font-family: Arial, sans-serif;

      display: flex;

      justify-content: center;

      align-items: center;
```

```
  height: 100vh;
  background: #f0f0f0;
}
.calculator {
  background: #fff;
  padding: 20px;
  border-radius: 12px;
  box-shadow: 0px 4px 10px rgba(0,0,0,0.2);
  width: 260px;
}
.display {
  width: 100%;
  height: 50px;
  font-size: 1.5rem;
  text-align: right;
  margin-bottom: 15px;
  padding: 5px;
  border: 2px solid #ccc;
  border-radius: 8px;
  background: #f9f9f9;
}
.buttons {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  gap: 10px;
}
button {
```

```
      padding: 15px;
      font-size: 1.2rem;
      border: none;
      border-radius: 8px;
      background: #007BFF;
      color: white;
      cursor: pointer;
      transition: background 0.2s ease;
    }
    button:hover {
      background: #0056b3;
    }
    .operator {
      background: #ff9800;
    }
    .operator:hover {
      background: #e68900;
    }
    .clear {
      background: #f44336;
    }
    .clear:hover {
      background: #d32f2f;
    }
  </style>
</head>
<body>
```

```
<div class="calculator">
  <input type="text" id="display" class="display" disabled>
  <div class="buttons">
    <button class="clear">C</button>
    <button class="operator">/</button>
    <button class="operator">*</button>
    <button class="operator">-</button>
    <button>7</button>
    <button>8</button>
    <button>9</button>
    <button class="operator">+</button>
    <button>4</button>
    <button>5</button>
    <button>6</button>
    <button>=</button>
    <button>1</button>
    <button>2</button>
    <button>3</button>
    <button>0</button>
  </div>
</div>

<script type="module">
  class Calculator {
    constructor(displayElement) {
      this.displayElement = displayElement;
      this.clear();
```

```
  }

  clear = () => {
    this.displayValue = "";
    this.updateDisplay();
  };

  append = (value) => {
    this.displayValue += value;
    this.updateDisplay();
  };

  calculate = () => {
    try {
      this.displayValue = eval(this.displayValue).toString();
    } catch {
      this.displayValue = "Error";
    }
    this.updateDisplay();
  };

  updateDisplay = () => {
    this.displayElement.value = this.displayValue;
  };
}

const display = document.getElementById("display");
```

```
    const buttons = document.querySelectorAll("button");
    const calc = new Calculator(display);

    buttons.forEach((btn) => {
      btn.addEventListener("click", () => {
        const value = btn.textContent;

        if (btn.classList.contains("clear")) {
          calc.clear();
        } else if (value === "=") {
          calc.calculate();
        } else {
          calc.append(value);
        }
      });
    });
  </script>
</body>
</html>
```

3. Read JSON data (from a local variable or file) and display it dynamically using HTML and JavaScript.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Read JSON Example</title>

</head>
<body>
  <h2>Read JSON Data</h2>
  <button onclick="loadFromVariable()">Load from Variable</button>
  <button onclick="loadFromFile()">Load from File</button>

  <div id="output"></div>
  <script>
    // Example JSON data (local variable)
    const jsonData = [
      { "name": "A Khan", "age": 44, "city": "Ranchi" },
      { "name": "Praveen", "age":21, "city": "Mangalore" },
      { "name": "Zainab", "age": 20, "city": "Pune" }
    ];

    // Function to render data dynamically
    function displayData(data) {
      const container = document.getElementById("output");
      container.innerHTML = ""; // Clear previous data

      data.forEach(item => {
        const card = document.createElement("div");
        card.className = "card";
        card.innerHTML = `<b>${item.name}</b><br>
                    Age: ${item.age}<br>
                    City: ${item.city}`;
        container.appendChild(card);
      });
    }

    // Load data from variable
    function loadFromVariable() {
      displayData(jsonData);
    }

    // Load data from JSON file
    function loadFromFile() {
      fetch("data.json") // <- local JSON file in same folder
        .then(response => response.json())
        .then(data => displayData(data))
        .catch(error => console.error("Error loading JSON:", error));
    }
  </script>
</body>
</html>
```

.....Data.Json.....

```
[
  { "name": "Praveen", "age": 20, "city": "Blore" },
  { "name": "Monisha", "age": 19, "city": "Chennai" }
]
```

4. Use Fetch API to call a public JSON API (e.g., weather, quotes, countries), parse the response, and display data dynamically using DOM manipulation.

```
<!DOCTYPE html>
<html>
<head>
  <title>Fetch API Example</title>
</head>
<body>
  <h1>Countries List (via Fetch API)</h1>
  <button id="load-btn">Load Countries</button>
  <ul id="country-list"></ul>

  <script>
    document.getElementById("load-btn").addEventListener("click", () => {
      // Fetch from the REST Countries API
      fetch("https://api.worldbank.org/v2/country?format=json&quot;)
        .then(response => {
          if (!response.ok) {
            throw new Error("Network response was not ok");
          }
          return response.json(); // Parse JSON
        })
        .then(data => {
          allcountry=data[1];
          const list = document.getElementById("country-list");
          list.innerHTML = ""; // Clear previous results

          // Loop through and display country names
          allcountry.forEach(country => {
            const li = document.createElement("li");
            li.textContent = country.name;
            list.appendChild(li);
          });
        })
        .catch(error => {
          console.error("Fetch error:", error);
          alert("Failed to load data. Check console for details.");
```

```
      });
   });
  </script>
</body>
</html>
```

5. Build a reusable student profile card component in React. Style it using Tailwind CSS with responsive layout and grid/flex utilities.

----- App.jsx----

```
import React from 'react'
import StudentProfileCard from './StudentProfileCard'


function App() {
 const socials = [
  {
    label: 'Twitter',
    url: 'https://twitter.com/janedoe&#39;,
    icon: <img src="" alt="Twitter" className="w-4 h-4" />
  },

 ]

 const handleAction = (action) => {
  console.log(`User clicked: ${action}`)
 }

 return (
  <div className="min-h-screen bg-slate-50 dark:bg-slate-900 flex items-center justify-center p-6">
    <StudentProfileCard
      name="Jane Doe"

avatar="https://images.unsplash.com/photo-1524504388940-b1c1722653e1?auto=format&fit=crop&w=256&q=80&quot;
      major="Computer Science"
      bio="Passionate about machine learning, open-source, and competitive programming."
      tags={["AI", "Open Source", "Speaker"]}
      socials={socials}
      onAction={handleAction}
    />
  </div>
 )
}

export default App
```

-----StudentProfileCard.jsx------

```
import React from 'react'

function StudentProfileCard({ name, avatar, major, bio, tags, socials, onAction }) {
  return (
    <div className="bg-white rounded-lg shadow-md p-6 max-w-sm">
      <div className="flex items-center space-x-3 mb-4">
        <img src={avatar} alt={name} className="w-12 h-12 rounded-full" />
        <div>
          <h2 className="text-lg font-bold">{name}</h2>
          <p className="text-sm text-gray-600">{major}</p>
        </div>
      </div>

      <p className="text-sm text-gray-700 mb-3">{bio}</p>

      <div className="flex gap-2 mb-3">
        {tags.map((tag, index) => (
          <span key={index} className="bg-blue-100 text-blue-800 px-2 py-1 rounded text-xs">
            {tag}
          </span>
        ))}
      </div>

      <div className="flex space-x-3 sm">
        {socials.map((social, index) => (
          <a key={index} href={social.url} className="text-gray-500 hover:text-blue-500">
            {social.icon}
          </a>
        ))}
      </div>

      <button
        onClick={() => onAction('Contact')}
        className="w-full bg-blue-500 text-white py-2 rounded hover:bg-blue-600"
      >
        Contact
      </button>
    </div>
  )
}

export default StudentProfileCard
```

6. Create a feedback using React form, include controlled components, form validations, and submit handling.

_____APP.JSX_____

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

function App() {
const [formData, setFormData] = useState({name: "", email: "", comments: "",});
const [errors, setErrors] = useState({});
const [submitted, setSubmitted] = useState(false);

// Handle form input change (controlled components)
const handleChange = (e) => {
const { name, value } = e.target;
setFormData((prev) => ({ ...prev, [name]: value }));
setErrors((prev) => ({ ...prev, [name]: "" }));
};

// Validation function
const validate = () => {
let newErrors = {};
if (!formData.name.trim()) newErrors.name = "Name is required";
if (!formData.email.trim()) {
newErrors.email = "Email is required";
} else if (!/\S+@\S+\.\S+/.test(formData.email)) {
newErrors.email = "Enter a valid email";
}
if (!formData.comments.trim()) newErrors.comments = "Comments are required";
return newErrors;
};

// Handle form submit
const handleSubmit = (e) => {
e.preventDefault();
const validationErrors = validate();
if (Object.keys(validationErrors).length > 0) {
setErrors(validationErrors);
} else {
console.log("Feedback submitted:", formData);
setSubmitted(true);
setFormData({ name: "", email: "", comments: "" });
}
};

return (
<div className="flex items-center justify-center min-h-screen bg-gray-100 p-4">
<div className="w-full max-w-md bg-white shadow-lg rounded-2xl p-6">
<h1 className="text-2xl font-semibold text-center mb-4 text-gray-700">  Feedback Form  </h1>

{submitted && (
      <div className="mb-4 p-3 text-green-700 bg-green-100 border border-green-300">
          Thank you for your feedback!
      </div>
   )}
```

```
<form onSubmit={handleSubmit} noValidate>
{/* Name */}
<div className="mb-4">
<label className="block text-gray-600 font-medium mb-1">
Name
</label>
<input type="text" name="name" value={formData.name} onChange={handleChange} />
{errors.name && (
<p className="text-red-500 text-sm mt-1">{errors.name}</p>
)}
</div>

{/* Email */}
<div className="mb-4">
<label className="block text-gray-600 font-medium mb-1">
Email
</label>
<input type="email" name="email" value={formData.email} onChange={handleChange}/>
{errors.email && (
<p className="text-red-500 text-sm mt-1">{errors.email}</p>
)}
</div>

{/* Comments */}
<div className="mb-4">
<label className="block text-gray-600 font-medium mb-1">
Comments
</label>
<textarea name="comments" value={formData.comments} onChange={handleChange} rows="4"/>
{errors.comments && (
<p className="text-red-500 text-sm mt-1">{errors.comments}</p>
)}
</div>

<button type="submit"> Submit Feedback </button>
</form>
</div>
</div>
);
}

export default App;
```

7. Build a small multi-page React app (e.g., Home, About, Contact) using React Router DOM with simple navigation and route rendering.

```
import { createRoot } from 'react-dom/client';
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
function Home() {
return <h1>Home Page</h1>;   }

function About() {
return <h1>About Page</h1>;  }

function Contact() {
return <h1>Contact Page</h1>; }

function App() {
  return (
    <BrowserRouter>      {/* Navigation */}
     <nav> <Link to="/">Home</Link> |{" "}
       <Link to="/about">About</Link> |{" "}
        <Link to="/contact">Contact</Link>       </nav>
      {/* Routes */}
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
      </Routes>
```

```
    </BrowserRouter>
  );
}
createRoot(document.getElementById('root')).render( <App />
);
```

8. Design a web page to register for an aspiring student to register for a program offered by the university using node.js and mysql database

(write The process of installation)

```
const express = require('express');
const mysql = require('mysql2');
const app = express();
app.use(express.json());
// Create MySQL Connection
const db = mysql.createConnection({
  host: 'localhost',
  user: 'root',       // your MySQL username
  password:",       // your MySQL password
  database: 'MyDB'
});
// Connect to MySQL
db.connect((err) => {
  if (err) {
    console.error('Database connection failed:', err);
    return;
  }
  console.log(' Connected to MySQL database.....');
  const i=125;
  const nm="Avinash";
  const eml="av@gmail.com";
  const sql = 'INSERT INTO users (id,uname, email) VALUES (?,?,?)';
  db.query(sql, [i,nm, eml], (err, result) => {
    if (err) throw err;
  console.log(' Record Saved.....');
  });
});
// Server
```

```
app.listen(3000, () => {
  console.log(' Server running on http://localhost:3000');
});
```

9. Design a web page to fetch the profile of a student from mongodb database using express.js

(write The process of installation)

```
//Lab Prog9 MongoDB based
// server.js
const express = require('express');
const mongoose = require('mongoose');
const app = express();
// Middleware
app.use(express.json());

//  1. Connect to MongoDB
mongoose.connect('mongodb://127.0.0.1:27017/studentDB', {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
.then(() => console.log(" MongoDB Connected"))
.catch(err => console.log(" Error: ", err));

//  2. Create Student Schema
const studentSchema = new mongoose.Schema({
  _id: Number,
  name: String,
  Program: String,
  Batch:String,
  Contact: Number
});

//  3. Student Model
const Student = mongoose.model('Student', studentSchema);

// 4. API: Get all students
app.get('/student', async (req, res) => {
  try {
    const stud = await Student.find();
    res.json(stud);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});

//  5. API: Get single student by roll
app.get('/student/:id', async (req, res) => {
```

```
    try {
      const stud = await Student.findOne({ _id: req.params.id });

      if (!stud) {
        return res.status(404).json({ message: "Student not found" });
      }

      res.json(stud);
    } catch (err) {
      res.status(500).json({ message: err.message });
    }
  });
  app.listen(3000, () => {
    console.log("Server running on http://localhost:3000&quot;);
  });
  //Sample Data
  {
    "_id": 12,
    "name": "Kumar",
    "Program": "BCA",
    "Batch": "2023-26",
    "Contact": 123455677
  }
```

10. Develop a backend REST API using Express.js to manage student records with endpoints for Create, Read, Update, and Delete operations.

- **server.js** file:

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

const app = express();

app.use(express.json());

let users = [

  { id: 1, name: 'Alice', email: 'alice@example.com' },

  { id: 2, name: 'Bob', email: 'bob@example.com' }

];

// CREATE

app.post('/api/users', (req, res) => {

  const newUser = {
```

```
    id: users.length + 1,
    name: req.body.name,
    email: req.body.email
  };
  users.push(newUser);
  res.status(201).json({ message: 'User created', user: newUser });
});
// READ (All)
app.get('/api/users', (req, res) => {
  res.json(users);
});
// READ (Single)
app.get('/api/users/:id', (req, res) => {
  const user = users.find(u => u.id === parseInt(req.params.id));
  if (!user) return res.status(404).json({ message: 'User not found' });
  res.json(user);
});

// UPDATE
app.put('/api/users/:id', (req, res) => {
  const user = users.find(u => u.id === parseInt(req.params.id));
  if (!user) return res.status(404).json({ message: 'User not found' });

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

  res.json({ message: 'User updated', user });
});
// DELETE
app.delete('/api/users/:id', (req, res) => {
  const index = users.findIndex(u => u.id === parseInt(req.params.id));
  if (index === -1) return res.status(404).json({ message: 'User not found' });

  const deleted = users.splice(index, 1);
```

```
  res.json({ message: 'User deleted', user: deleted });
});
const PORT = 3000;
app.listen(PORT, () => console.log(`Server running on http://localhost:${PORT}`));
```

14. Write a simple function to return current server time.

-----lab 14 ---

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

// Create a server
const server = http.createServer((req, res) => {
  if (req.url === '/time') {
    const currentTime = new Date().toLocaleString(); // Get local time
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end(`Current Server Time: ${currentTime}`);
  } else {
    res.writeHead(404, { 'Content-Type': 'text/plain' });
    res.end('Endpoint not found. Use /time');
  }
});

// Listen on port 3000
server.listen(3000, () => {
  console.log('✅ Server running at http://localhost:3000&#39;);
});
```