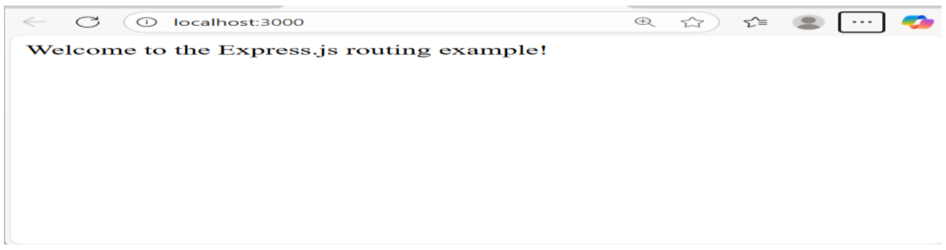
1. Express JS – Routing, HTTP Methods, Middleware.

a. Write a program to define a route, Handling Routes, Route Parameters, Query Parameters and URL building.

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

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

// PORT
const PORT = 3000;

// Home route
app.get('/', (req, res) => {
  res.send('Welcome to the Express.js routing example!');
});

// Route with route parameters
app.get('/user/:id', (req, res) => {
  const userId = req.params.id;
  res.send(`User ID from route parameter is: ${userId}`);
});

// Route with multiple route parameters
app.get('/user/:userId/book/:bookId', (req, res) => {
  const { userId, bookId } = req.params;
  res.send(`User ID: ${userId}, Book ID: ${bookId}`);
});

// Route with query parameters
app.get('/search', (req, res) => {
  const { keyword, limit } = req.query;
  res.send(`Search keyword: ${keyword}, Limit: ${limit}`);
});

// POST route to demonstrate body parsing
app.post('/user', (req, res) => {
  const { name, age } = req.body;
  res.send(`Received user data: Name = ${name}, Age = ${age}`);
});
// URL building example
app.get('/build-url', (req, res) => {
  const userId = 42;
  const bookId = 7;
  const builtUrl = `/user/${userId}/book/${bookId}`;
  res.send(`Dynamically built URL: ${builtUrl}`);
});

// Catch-all route for undefined paths
app.use((req, res) => {
  res.status(404).send('404 Not Found');
});

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

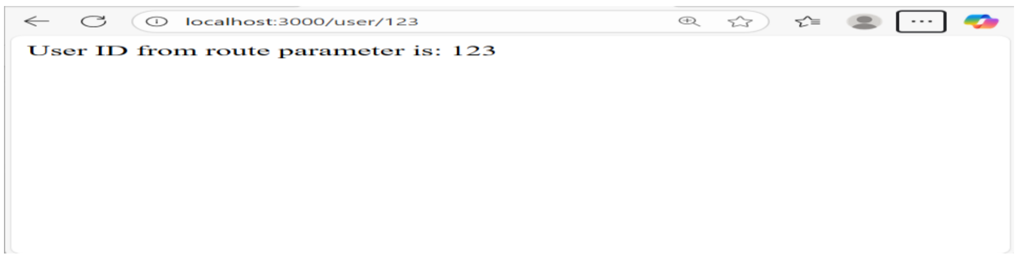
OUTPUT:

```
PS D:\fsd> node  exercise1.js
Server running on http://localhost:3000
```



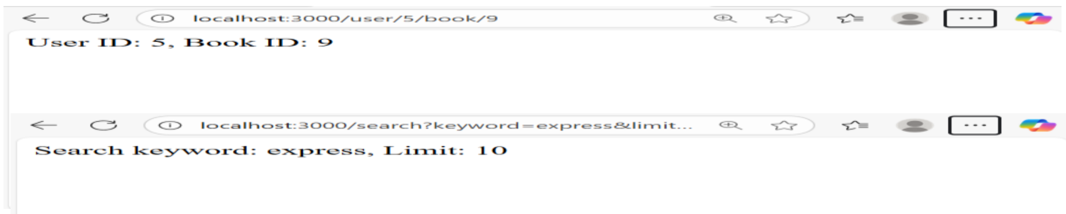
localhost:3000

```
Welcome to the Express.js routing example!
```



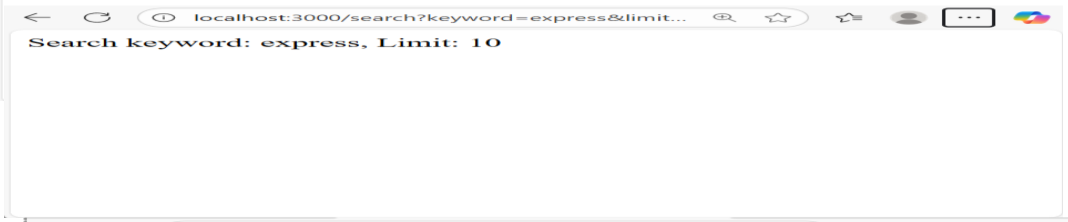
localhost:3000/user/123

```
User ID from route parameter is: 123
```



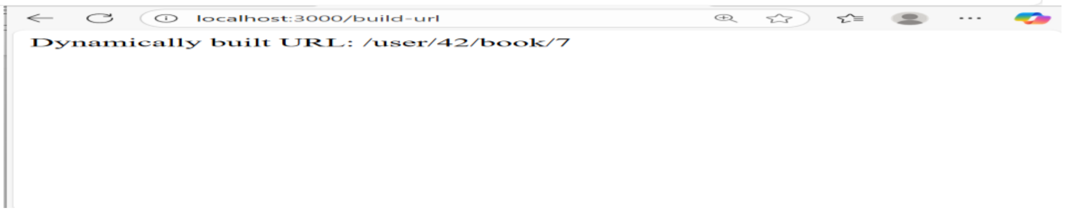
localhost:3000/user/5/book/9

```
User ID: 5, Book ID: 9
```



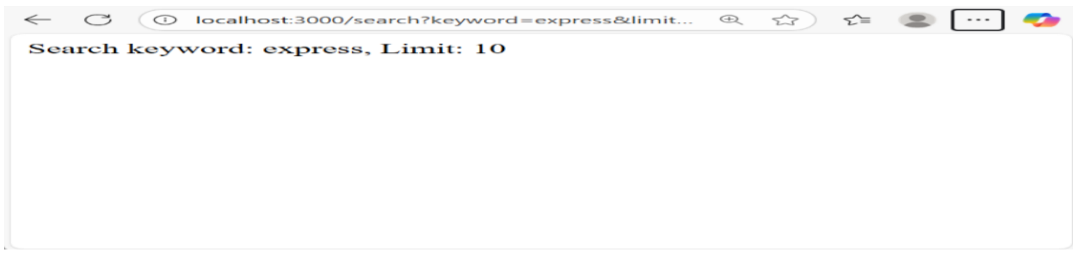
localhost:3000/search?keyword=express&limit...

```
Search keyword: express, Limit: 10
```



localhost:3000/build-url

```
Dynamically built URL: /user/42/book/7
```



localhost:3000/search?keyword=express&limit...

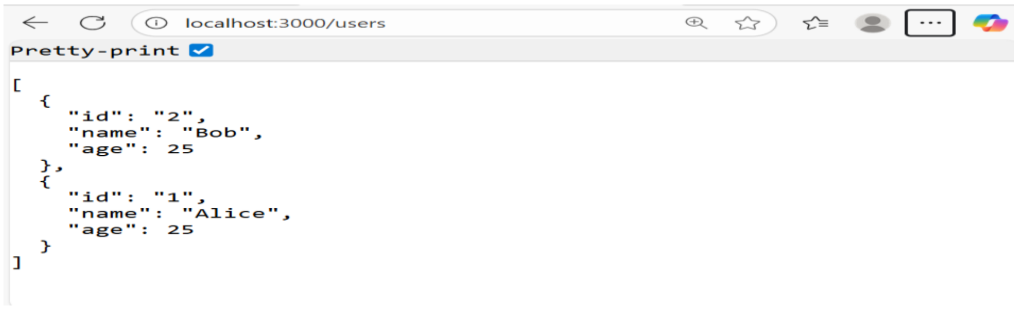
```
Search keyword: express, Limit: 10
```

b. Write a program to accept data, retrieve data and delete a specified resource using http methods.

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

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

// In-memory array to simulate a database
let users = [];

// POST: Add new user
app.post('/users', (req, res) => {
  const { id, name, age } = req.body;
  if (!id || !name || !age) {
    return res.status(400).send("Missing id, name, or age.");
  }
  users.push({ id, name, age });
  res.status(201).send(`User added: ${name}`);
});

// GET: Retrieve all users
app.get('/users', (req, res) => {
  res.status(200).json(users);
});

// DELETE: Delete user by ID
app.delete('/users/:id', (req, res) => {
  const userId = req.params.id;
  const originalLength = users.length;
  users = users.filter(user => user.id !== userId);

  if (users.length === originalLength) {
    return res.status(404).send(`User with ID ${userId} not found.`);
  }
  res.send(`User with ID ${userId} deleted.`);
});
// Start the server
const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

OUTPUT:

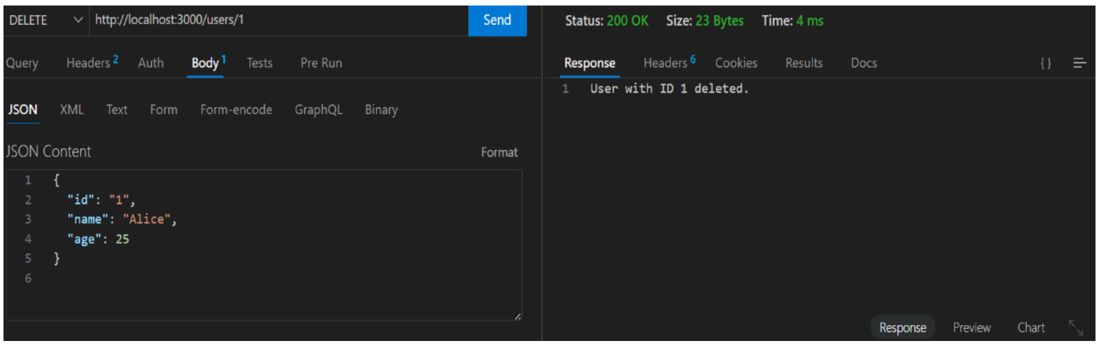
```
PS D:\fsd> node  exercise1b.js
Server running on http://localhost:3000
```



localhost:3000/users

Pretty-print ✓

```
[
  {
    "id": "2",
    "name": "Bob",
    "age": 25
  },
  {
    "id": "1",
    "name": "Alice",
    "age": 25
  }
]
```



DELETE http://localhost:3000/users/1 Send

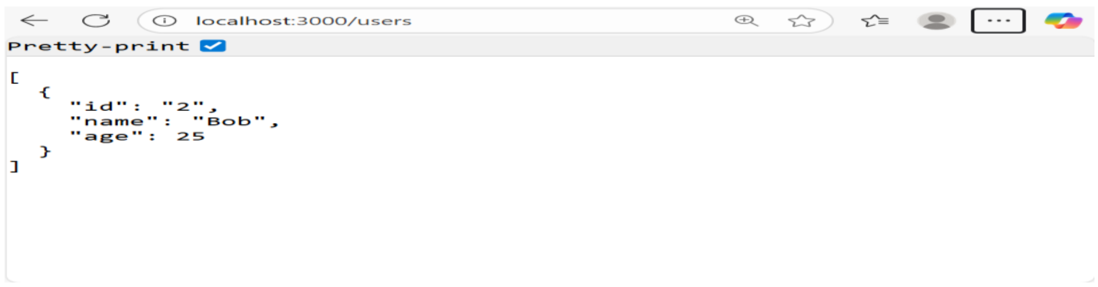
Status: 200 OK Size: 23 Bytes Time: 4 ms

JSON Content

```
{
  "id": "1",
  "name": "Alice",
  "age": 25
}
```

Response:

```
User with ID 1 deleted.
```



localhost:3000/users

Pretty-print ✓

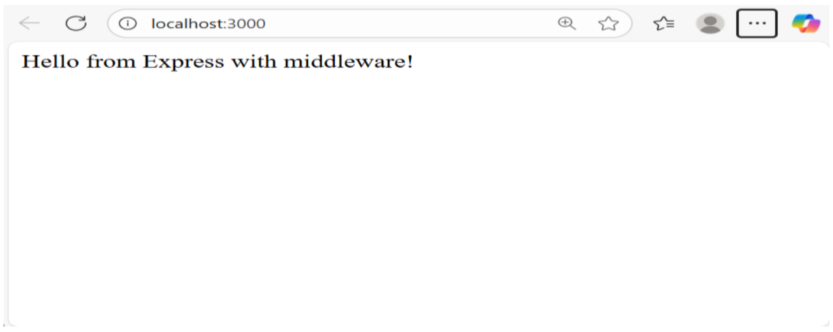
```
[
  {
    "id": "2",
    "name": "Bob",
    "age": 25
  }
]
```

1.c. **Write a program to show the working of middleware**.

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

// Built-in middleware to parse JSON
app.use(express.json());

// Custom middleware: Logger
app.use((req, res, next) => {
  console.log(`[${new Date().toISOString()}] ${req.method} ${req.url}`);
  next(); // Pass control to the next middleware/route
});

// Route
app.get('/', (req, res) => {
  res.send('Hello from Express with middleware!');
});

// POST route to test JSON body
app.post('/user', (req, res) => {
  const { name } = req.body;
  res.send(`Received user name: ${name}`);
});

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

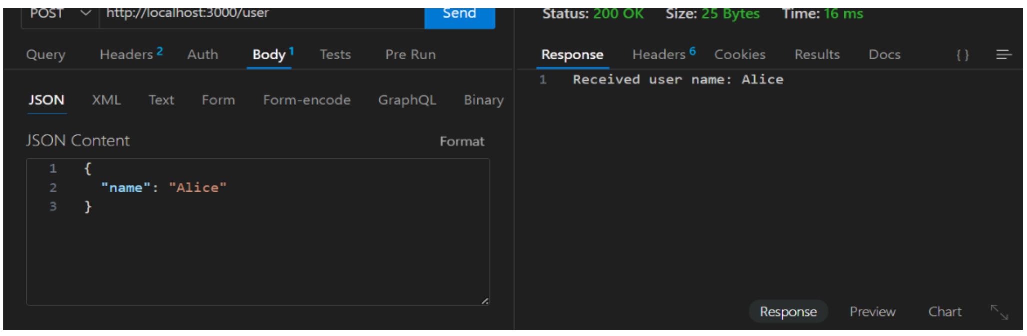
OUTPUT:

```
PS D:\fsd> node  exercise1c.js
Server running at http://localhost:3000
good
Request Type: GET


PS D:\fsd> node  exercise1c.js
Server running at http://localhost:3000
good
Request Type: GET
good
Request Type: POST
```



localhost:3000

Hello from Express with middleware!



POST http://localhost:3000/user Send

Status: 200 OK Size: 25 Bytes Time: 16 ms

JSON Content

```
{
  "name": "Alice"
}
```

```
Received user name: Alice
```

2. Express JS – Templating, Form Data

a. Write a program using templating engine.

Step 1: Initialize the Project

```
mkdir express-ejs-template
cd express-ejs-template
npm init
npm install express ejs
```

Project Structure:

```
express-ejs-template/
 ├── views/
|   └── profile.ejs
 ├── public/
|   └── style.css
 ├── app.js
```

Step 2: views/profile.ejs (EJS Template)

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title><%= name %>'s Profile</title>
   <link rel="stylesheet" href="/style.css">
</head>
<body>
   <h1>User Profile</h1>
   <p><strong>Name:</strong> <%= name %></p>
   <p><strong>Age:</strong> <%= age %></p>
   <p><strong>City:</strong> <%= city %></p>
</body>
</html>
```

Optional: public/style.css

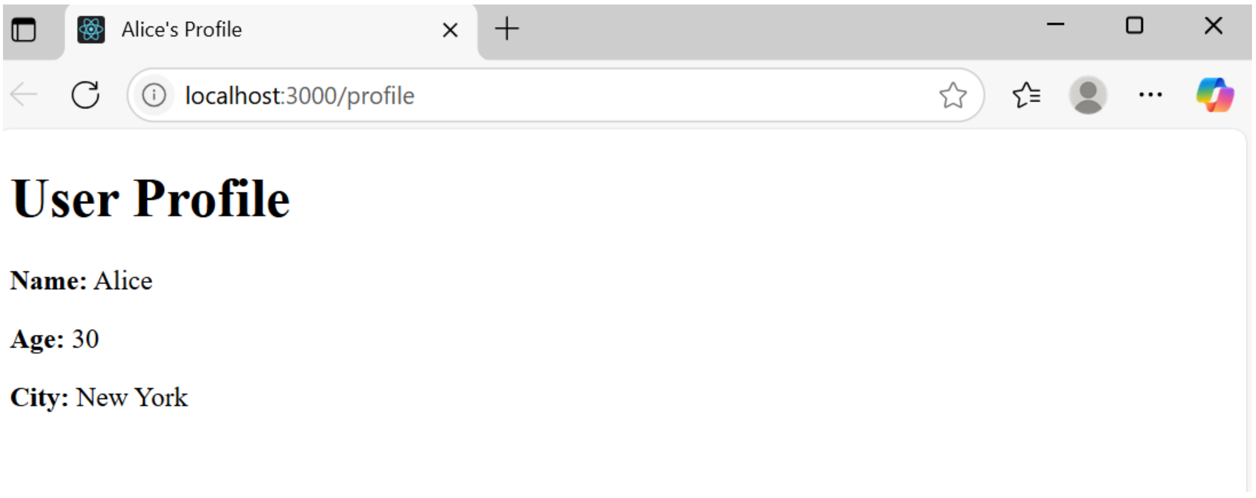
```
body {
   font-family: Arial, sans-serif;
   margin: 40px;
}
h1 {
   color: #2c3e50;
}
```

Step 3: app.js (Express Server with EJS)

```
const express = require('express');
const app = express();
const port = 3000;
// Set EJS as templating engine
app.set('view engine', 'ejs');

// Serve static files from "public"
app.use(express.static('public'));

// Route to render user profile
app.get('/profile', (req, res) => {
  const user = {
    name: 'Alice',
    age: 30,
    city: 'New York'
  };
  res.render('profile', user);
});
app.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}/profile`);
});
```

OUTPUT:

```
PS D:\fsd> node  exercise2.js
Server running at http://localhost:3000
Then visit: http://localhost:3000/profile
```



User Profile

Name: Alice

Age: 30

City: New York

b. Write a program to work with form data

Forms are an integral part of the web. Almost every website we visit offers us forms that submit or fetch some information for us.

Step 1: Set Up the Project

```
mkdir express-ejs-form
cd express-ejs-form
npm init -y
npm install express ejs
```

Folder Structure

```
express-ejs-form/
 ├── views/
|   ├── form.ejs
|   └── result.ejs
 ├── app.js
```

Step 2: Create Views

views/form.ejs

```
html
CopyEdit
<!DOCTYPE html>
<html>
<head>
  <title>User Form</title>
</head>
<body>
  <h1>User Information Form</h1>
  <form action="/submit" method="POST">
    <label>Name:</label>
    <input type="text" name="name" required><br><br>

    <label>Email:</label>
    <input type="email" name="email" required><br><br>

    <button type="submit">Submit</button>
  </form>
</body>
</html>
```

views/result.ejs

```
<!DOCTYPE html>
<html>
<head>
  <title>Form Result</title>
</head>
<body>
  <h1>Form Submitted</h1>
  <p><strong>Name:</strong> <%= name %></p>
  <p><strong>Email:</strong> <%= email %></p>
</body>
</html>
```

Step 3: app.js

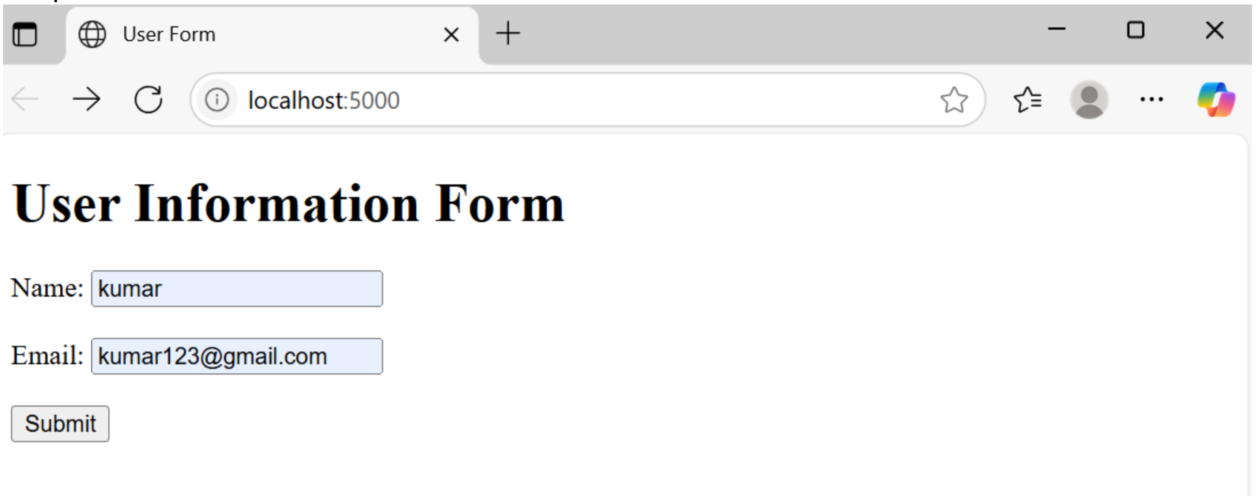
```
        const express = require('express');
        const app = express();
        const port = 5000;

        // Middleware to parse form data
        app.use(express.urlencoded({ extended: true }));

        // Set EJS as the templating engine
        app.set('view engine', 'ejs');
        app.set('views', './views');

        // GET route to render the form
        app.get('/', (req, res) => {
          res.render('form');
        });

        // POST route to handle form submission
        app.post('/submit', (req, res) => {
          const { name, email } = req.body;
          res.render('result', { name, email });
        });

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

OUTPUT:

```
PS D:\fsd> node  exercise2b.js
Server running at http://localhost:5000
```

Then open: http://localhost:5000

Output:



User Form

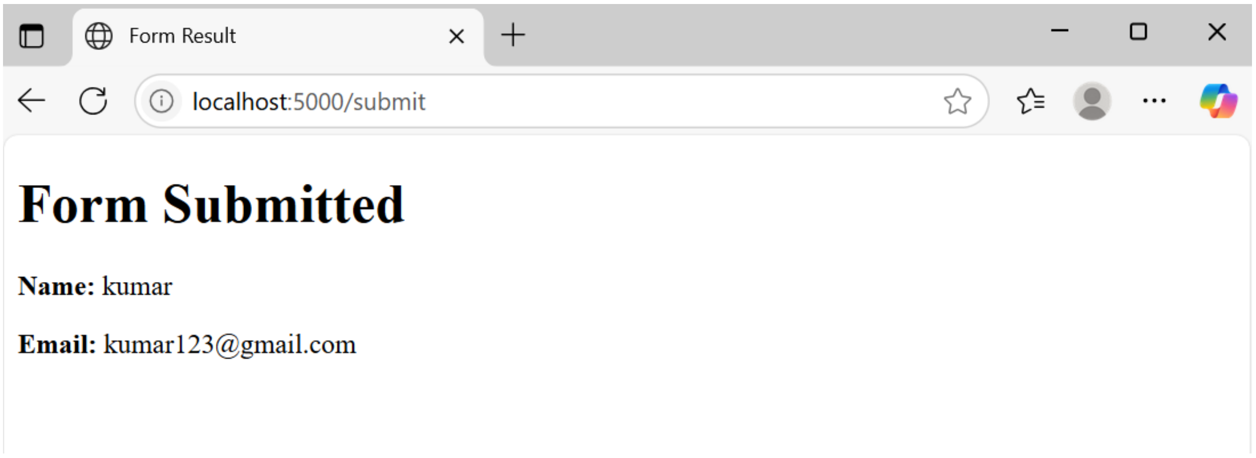
localhost:5000

User Information Form

Name: kumar

Email: kumar123@gmail.com

Submit



Form Result

localhost:5000/submit

Form Submitted

Name: kumar

Email: kumar123@gmail.com

3. Express JS – Cookies, Sessions, Authentication

a. Write a program for session management using cookies

1. Install Express and cookie-parser

```
npm init          # initialize your project
npm install express cookie-parser
```

2. Create the App (app.js)

```
// Import required modules
const express = require('express');
const cookieParser = require('cookie-parser');

// Create Express app
const app = express();
const PORT = 3000;
// Use cookie-parser middleware
app.use(cookieParser());
// Route 1: Set a cookie
app.get('/set-cookie', (req, res) => {
  res.cookie('username', 'Kumar, {
    maxAge: 60000, // cookie valid for 60 seconds
    httpOnly: true // cookie not accessible via JavaScript
  });
  res.send('Cookie has been set');
});

// Route 2: Get the cookie
app.get('/get-cookie', (req, res) => {
  const username = req.cookies.username;

  if (username) {
    res.send(` Cookie value: ${username}`);
  } else {
    res.send(' No cookie found');
  }
});

// Route 3: Clear the cookie
app.get('/clear-cookie', (req, res) => {
  res.clearCookie('username');
  res.send('Cookie has been cleared');
});

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

OUTPUT:

http://localhost:3000/set-cookie Sets a cookie named username with value Kumar

http://localhost:3000/get-cookie Retrieves the cookie value

http://localhost:3000/clear-cookie Deletes the cookie

b. Write a program for session management using sessions.

1. **Create project & install dependencies**.

```
mkdir express-sessions
cd express-sessions

npm init
npm install express express-session
```

2. **Create the App(app.js)**

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

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



const app = express();



// Middleware to parse POST data (if needed)

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



// Session configuration

app.use(

  session({

    secret: 'my_secret_key', // should be a long, random string in production

    resave: false,          // don't save session if unmodified

    saveUninitialized: false,// don't create session until something stored

    cookie: {

      maxAge: 1000 * 60 * 5  // session expires after 5 minutes
    }
  })
);


// Route to set a session variable
app.get('/set-session', (req, res) => {
  req.session.username = 'JohnDoe';
  res.send('Session data set: username = JohnDoe');
});


// Route to get the session variable
app.get('/get-session', (req, res) => {
  if (req.session.username) {
    res.send(`Hello ${req.session.username}, your session is active.`);
  } else {
    res.send('No session data found.');
  }
});


// Route to destroy session
app.get('/destroy-session', (req, res) => {
  req.session.destroy(err => {
    if (err) {
      return res.send('Error destroying session');
    }
    res.clearCookie('connect.sid'); // clear the cookie
    res.send('Session destroyed successfully.');
  });
});

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

OUTPUT:

- http://localhost:3000/set-session → sets the session
- http://localhost:3000/get-session → retrieves the session
- http://localhost:3000/destroy-session → destroys the session

c. Write a program for user authentication

1) Create project & install dependencies.

```
mkdir express-auth-sessions

cd express-auth-sessions

npm init -y

npm install express express-session
```

2) Create app.js

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

const app = express();

// Parse form/json bodies
app.use(express.urlencoded({ extended: true }));
app.use(express.json());

// Session middleware (MemoryStore — fine for dev only)
app.use(session({
  secret: 'replace_this_with_a_long_random_string',
  resave: false,
  saveUninitialized: false,
  cookie: {
    maxAge: 1000 * 60 * 10, // 10 minutes
    // secure: true, // enable only when serving over HTTPS
    httpOnly: true,
    sameSite: 'lax',
  }
}));

// Dummy users (for demo)
const users = [
  { username: 'admin', password: '12345' },
  { username: 'user',  password: 'password' }
];

// Small helper: protect routes
function ensureAuth(req, res, next) {
  if (req.session.user) return next();
  return res.status(401).send('Not authenticated. <a href="/login">Login</a>');
}

// Login form (GET)
app.get('/login', (req, res) => {
  res.send(`
    <h2>Login</h2>
    <form method="POST" action="/login">
      <input name="username" placeholder="Username" required />
      <br/><br/>
      <input type="password" name="password" placeholder="Password" required />
      <br/><br/>
      <button>Login</button>
    </form>
  `);
});

// Login handler (POST)
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  const match = users.find(u => u.username === username && u.password === password);
  if (!match) return res.status(401).send('Invalid credentials. <a href="/login">Try again</a>');
  req.session.user = { username: match.username };
  res.redirect('/dashboard');
});

// Protected page
app.get('/dashboard', ensureAuth, (req, res) => {
  res.send(`Hello ${req.session.user.username}! <a href="/me">Who am I?</a> | <a
href="/logout">Logout</a>`);
});

// Who am I (reads session)
app.get('/me', ensureAuth, (req, res) => {
  res.json({ sessionUser: req.session.user, sessionID: req.sessionID });
});

// Logout (destroy session)
app.get('/logout', (req, res) => {
  req.session.destroy(err => {
    if (err) return res.status(500).send('Error logging out.');
    res.clearCookie('connect.sid');

    res.redirect('/login');

  });

});


app.listen(3000, () => console.log('http://localhost:3000'));
```

OUTPUT:

Open http://localhost:3000/login

Login with admin / 12345 (or user / password)

Visit http://localhost:3000/dashboard

Try http://localhost:3000/me

Logout at http://localhost:3000/logout

4. Express JS – Database, RESTful APIs

a. Write a program to connect MongoDB database using Mangoose and perform CRUD operations.

Steps

1. Install dependencies:
   ```
   npm init -y
   npm install express mongoose ejs
   ```
2. Create this project structure:
   ```
   project/
   |— server.js
   |— views/
     |— index.ejs
     |— edit.ejs
   ```

Server.js

```
const express = require("express");
const mongoose = require("mongoose");
const path = require("path");

const app = express();

// Middleware
app.use(express.urlencoded({ extended: true })); // Parse form data
app.set("view engine", "ejs");
app.set("views", path.join(__dirname, "views"));

// MongoDB connection
const DB_URL = "mongodb+srv://fsda:fsda@cluster0.mxvsjjm.mongodb.net/fsda ";

mongoose
  .connect(DB_URL, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => {
    console.log(" MongoDB connected successfully");

    // Start server only if DB connected
    app.listen(3000, () => {
      console.log(" Server running at http://localhost:3000");
    });
  })
  .catch((err) => {
    console.error(" MongoDB connection failed:", err.message);
    process.exit(1); // Stop app if DB not connected
  });
```

```
// Schema + Model
const userSchema = new mongoose.Schema({
  name: String,
  email: String,
  age: Number,
});
const User = mongoose.model("User", userSchema);

// ---------------- ROUTES ----------------

// Home → List users + Add form
app.get("/", async (req, res) => {
  const users = await User.find();
  res.render("index", { users });
});

// CREATE
app.post("/users", async (req, res) => {
  await User.create(req.body);
  res.redirect("/");
});

// EDIT form
app.get("/users/edit/:id", async (req, res) => {
  const user = await User.findById(req.params.id);
  res.render("edit", { user });
});

// UPDATE
app.post("/users/update/:id", async (req, res) => {
  await User.findByIdAndUpdate(req.params.id, req.body);
  res.redirect("/");
});

// DELETE
app.post("/users/delete/:id", async (req, res) => {
  await User.findByIdAndDelete(req.params.id);
  res.redirect("/");
});
```

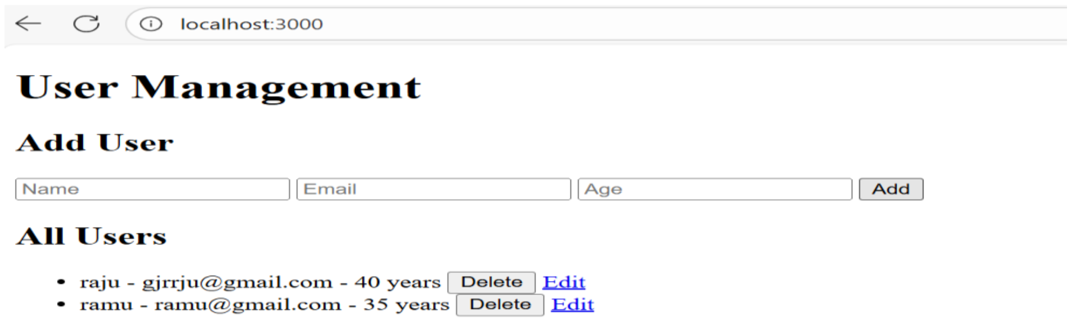
views/index.ejs

```
<!DOCTYPE html>
<html>
<head>
  <title>CRUD with Forms</title>
</head>
<body>
  <h1>User Management</h1>
  <h2>Add User</h2>
  <form action="/users" method="POST">
    <input type="text" name="name" placeholder="Name" required />
    <input type="email" name="email" placeholder="Email" required />
    <input type="number" name="age" placeholder="Age" required />
    <button type="submit">Add</button>
  </form>
  <h2>All Users</h2>
  <ul>
    <% users.forEach(user => { %>
      <li>
        <%= user.name %> - <%= user.email %> - <%= user.age %> years
        <form action="/users/delete/<%= user._id %>" method="POST" style="display:inline;">
          <button type="submit">Delete</button>
        </form>
        <a href="/users/edit/<%= user._id %>">Edit</a>
      </li>
    <% }) %>
  </ul>
</body>
</html>
```

views/edit.ejs

```
<!DOCTYPE html>
<html>
<head>
  <title>Edit User</title></head>
<body>
  <h1>Edit User</h1>
  <form action="/users/update/<%= user._id %>" method="POST">
    <input type="text" name="name" value="<%= user.name %>" required />
    <input type="email" name="email" value="<%= user.email %>" required />
    <input type="number" name="age" value="<%= user.age %>" required />
    <button type="submit">Update</button>
  </form>
  <a href="/">Back</a>
</body></html>
```

Output:



localhost:3000

User Management

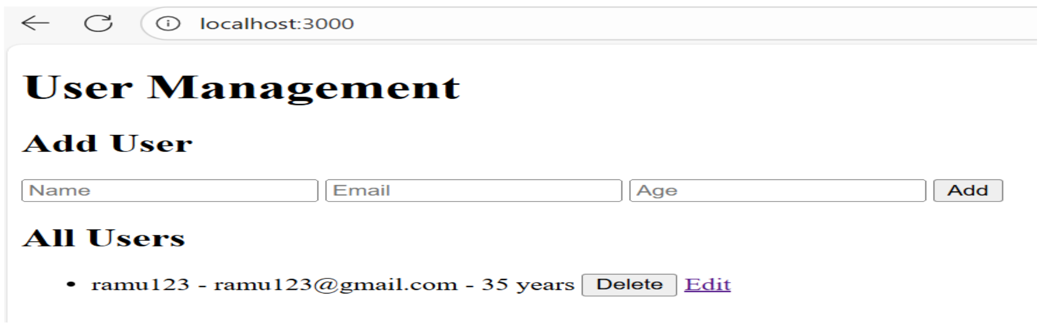
Add User

Name | Email | Age | Add

All Users

- raju - gjrrju@gmail.com - 40 years Delete Edit
- ramu - ramu@gmail.com - 35 years Delete Edit

Add user



localhost:3000

User Management

Add User

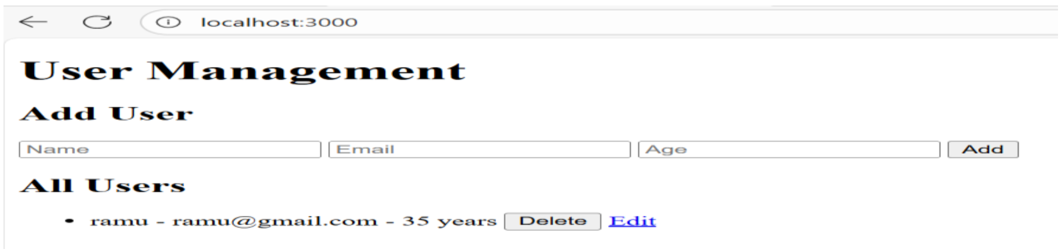
Name | Email | Age | Add

All Users

- ramu123 - ramu123@gmail.com - 35 years Delete Edit

Delete

User



localhost:3000

User Management

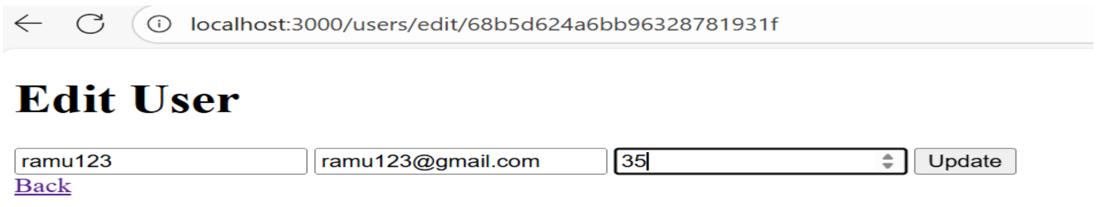
Add User

Name | Email | Age | Add

All Users

- ramu - ramu@gmail.com - 35 years Delete Edit

Update user



localhost:3000/users/edit/68b5d624a6bb96328781931f

Edit User

ramu123 | ramu123@gmail.com | 35 | Update

Back

5. ReactJS – Render HTML, JSX, Components – function & Class

a. Rendering HTML to a Web Page:

React renders content into a designated DOM element, typically a div with an id like "root" in your index.html file.

Steps:

1. Create folder
2. npm create vite@latest my-app
   cd my-app
   npm install
   npm run dev

Default Structure (after creating app)

Vite (React + JS):

```
my-app/
|— node_modules/       → installed packages
|— public/             → static assets (images, icons, etc.)
|— src/                → application source code
|   ├— App.jsx         → root component
|   ├— main.jsx        → entry point (renders App)
|— index.html          → main HTML template
|— package.json        → project dependencies & scripts
|— vite.config.js      → Vite configuration
```

<u>App.jsx</u>

```
function App() {
  return (
    <div className="App">
      <h1>Hello World!</h1>
    </div>
  );
}

export default App;
```

main.jsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
     <App />
   </StrictMode>
  )
```

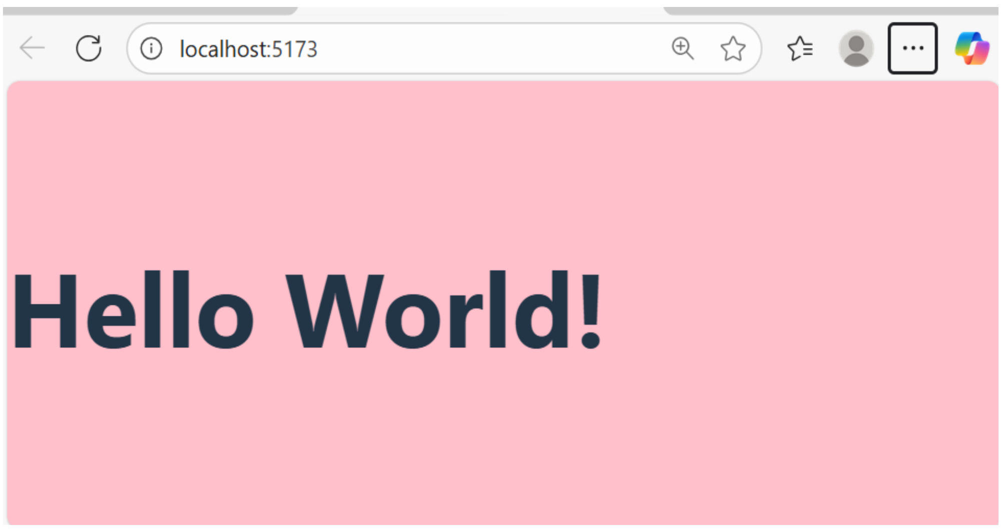
index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>React App</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

OUTPUT:

```
PS D:\fsd\exp5\firstreact> npm run dev
```

http://localhost:5173/



Hello World!

b. Writing Markup with JSX:

JSX allows you to write HTML-like syntax directly within your JavaScript code.

// src/App.jsx

```
import './App.css'
import React from 'react';
import Greeting from './components/Greetings';
import Car from './Vehicle';

function App() {
  const name = "World";
  return (
    <div>
      <h2>JSX Example</h2>
      <p>Hello, {name}!</p>
      <Greeting />
      <Car />
    </div>
  );
}
export default App;
```

main.jsx

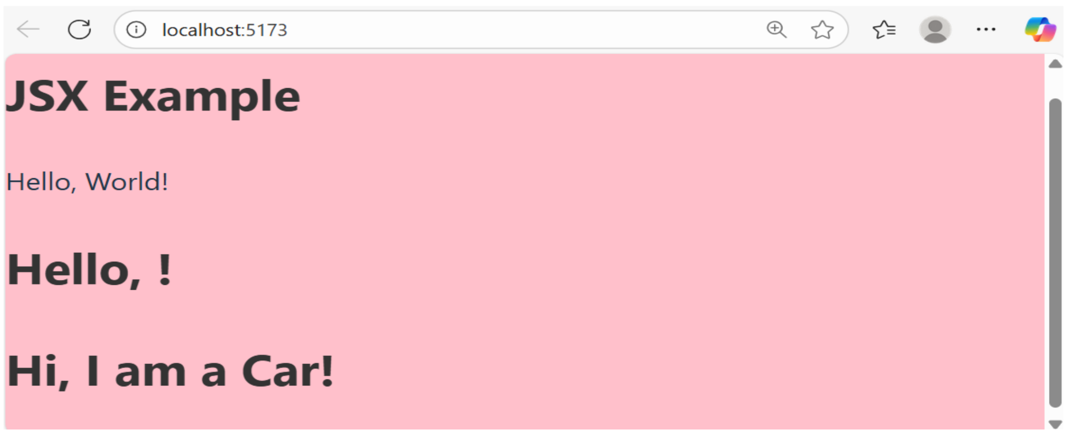
```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
      <App />
    </StrictMode>
  )
```

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>React App</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

OUTPUT:

```
PS D:\fsd\exp5\firstreact> npm run dev

http://localhost:5173/
```



JSX Example

Hello, World!

Hello, !

Hi, I am a Car!

c. Creating and Nesting Components:

1. **Functional Components:**

Functional components are JavaScript functions that return JSX.

// src/components/Greeting.jsx

```
import React from 'react';
function Greeting() {
  return <h3>Hello</h3>;
}
export default Greeting;
```

2. Class Components:

- Class components are ES6 classes that extend React.Component
- Component is the base class for the React components defined as JavaScript classes. that return JSX.

// src/components/Vehicle.jsx

```
import React from 'react';
class Car extends React.Component {
    render() {
        return <h2>Hi, I am a Car!</h2>;
    }
}
export default Car;
```

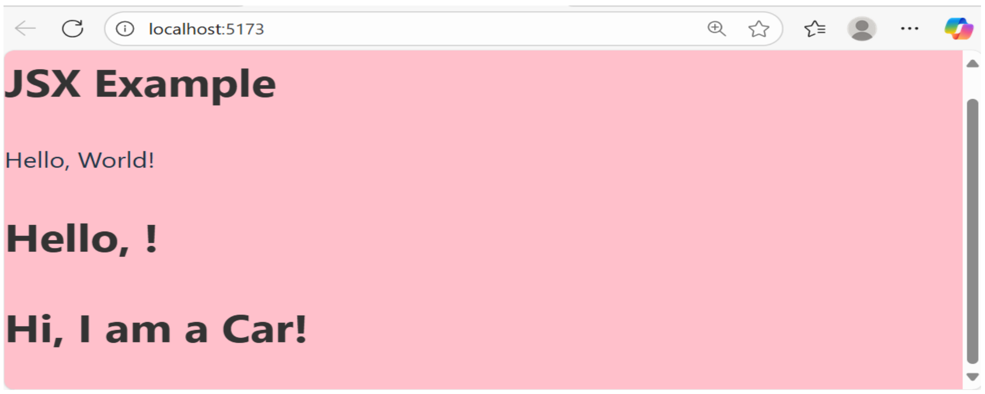
main.jsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'
createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

OUTPUT:

```
PS D:\fsd\exp5\firstreact> npm run dev
```

http://localhost:5173/



JSX Example

Hello, World!

Hello, !

Hi, I am a Car!

6. ReactJS – Props and States, Styles, Respond to Events

a. Write a program to work with props and states.

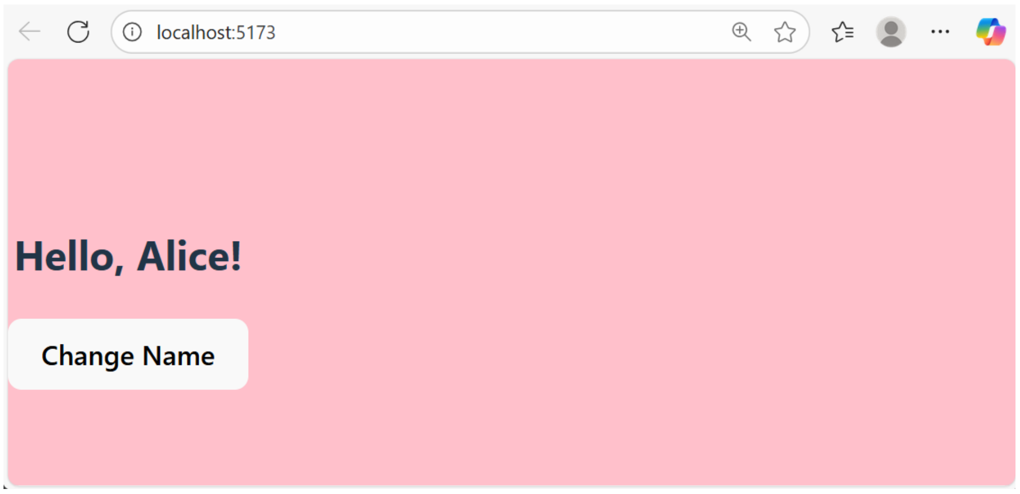
src/Greeting.jsx

```
import React from "react";

export default function Greeting({ userName }) {
  return <h2>Hello, {userName}!</h2>;
}
```

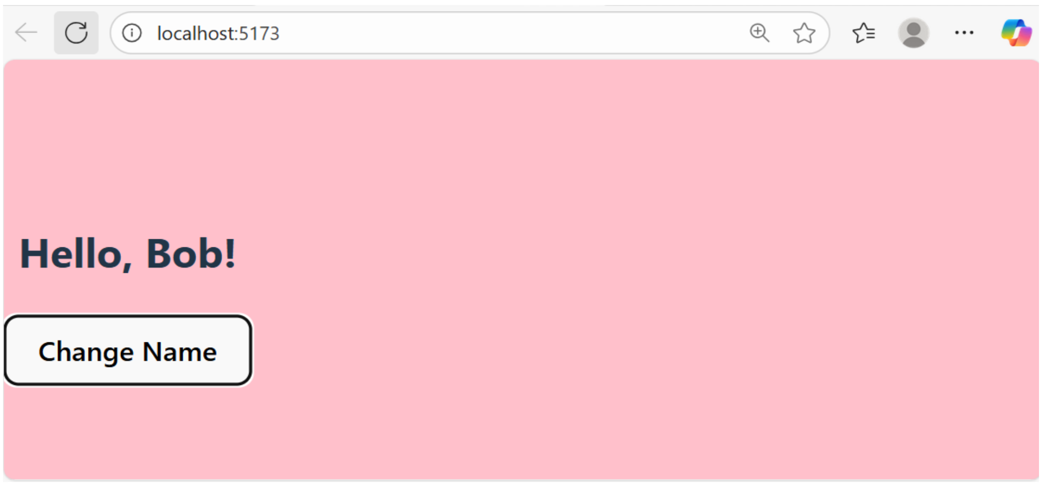
src/App.jsx

```
import React, { useState } from "react";
import Greeting from "./Greeting";

export default function App() {
  const [name, setName] = useState("Alice");

  return (
    <div style={{ textAlign: "center", marginTop: "40px" }}>
      <Greeting userName={name} />
      <button onClick={() => setName("Bob")}>Change Name</button>
    </div>
  );
}
```

OUTPUT:


localhost:5173

Hello, Alice!


localhost:5173

Hello, Bob!

b) Program to Add **Styles** (CSS & Sass) and Display Data

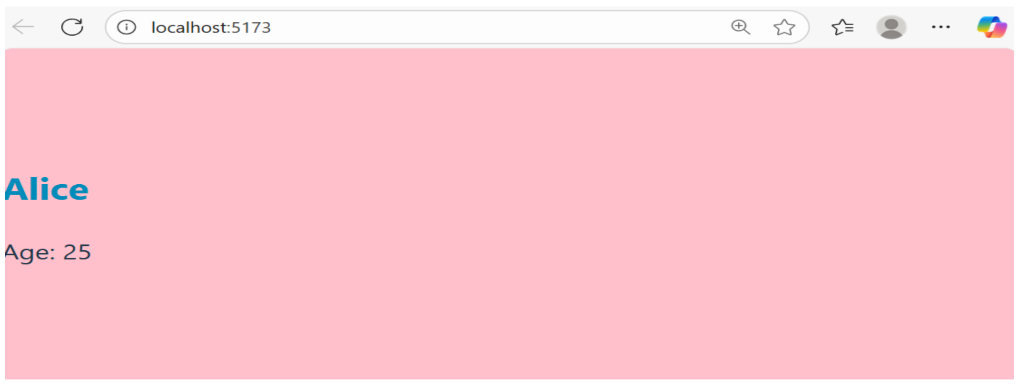
src/App.css

```
.card {
  background: #f0f8ff;
  border: 2px solid #008cba;
  border-radius: 12px;
  padding: 20px;
  width: 250px;
  text-align: center;
  margin: 20px auto;
}
.title {
  color: #008cba;
  font-size: 1.4rem;
}
```

src/App.jsx

```
import React from "react";
import "./App.css";

export default function App() {
  const user = { name: "Alice", age: 25 };

  return (
    <div className="card">
      <h2 className="title">{user.name}</h2>
      <p>Age: {user.age}</p>
    </div>
  );
}
```

OUTPUT:



Alice

Age: 25

c) Program Responding to Events

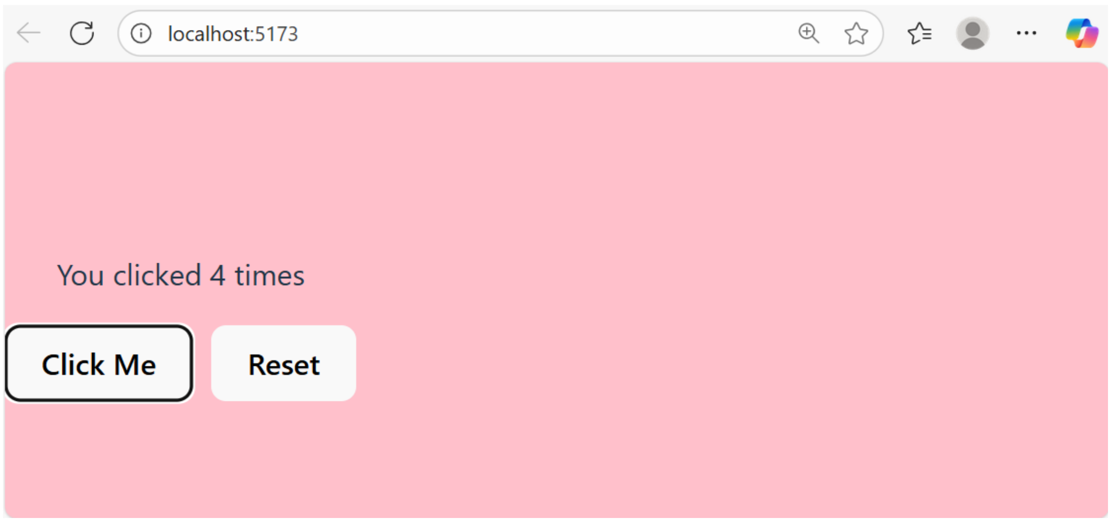
src/ClickCounter.jsx

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

export default function ClickCounter() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount(count + 1);
  }

  function handleReset() {
    setCount(0);
  }

  return (
    <div style={{ textAlign: "center", marginTop: "40px" }}>
      <p>You clicked {count} times</p>
      <button onClick={handleClick}>Click Me</button>
      <button onClick={handleReset} style={{ marginLeft: "10px" }}>
        Reset
      </button>
    </div>
  );
}
```

src/App.jsx

```
import React from "react";
import ClickCounter from "./ClickCounter";

export default function App() {
  return <ClickCounter />;
}
```

OUTPUT:

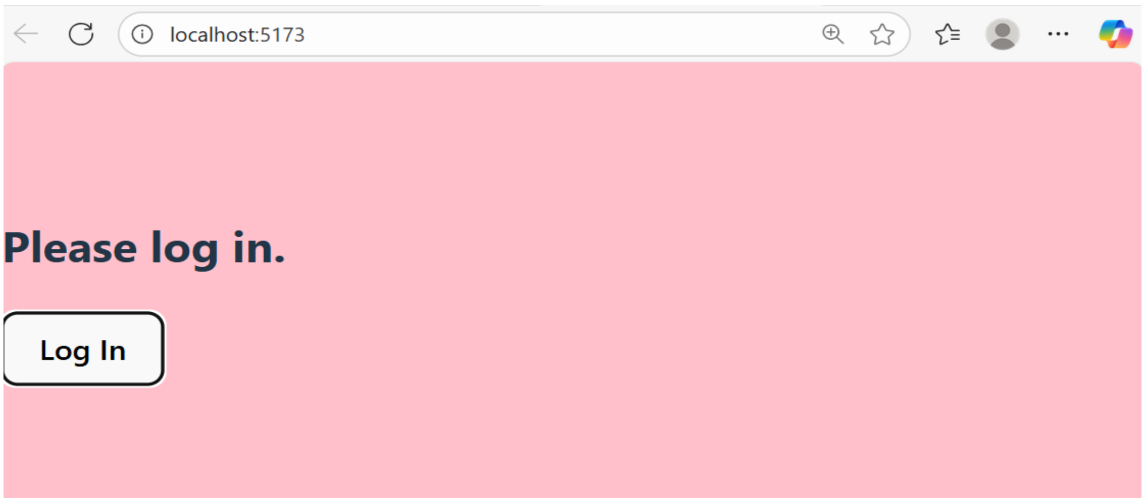

You clicked 4 times

7. ReactJS – Conditional Rendering, Rendering Lists, React Forms

a. Write a program for conditional rendering.

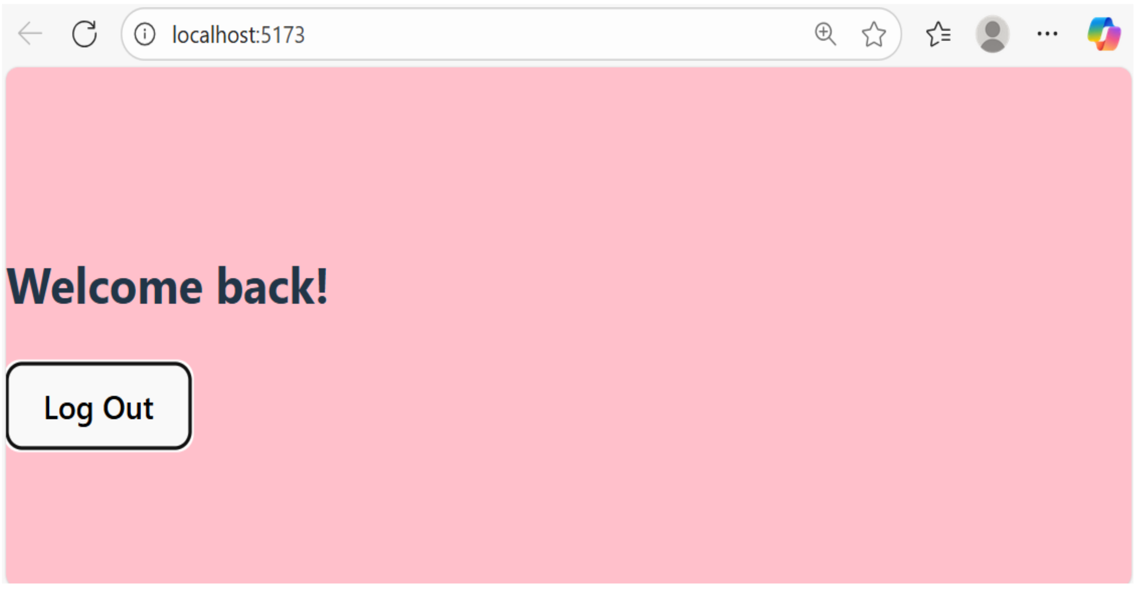
App.jsx

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

export default function App() {

  const [isLoggedIn, setIsLoggedIn] = useState(false);

  return (

    <div>

      {/* Show one message if logged in, another if not */}

      {isLoggedIn ? <h2>Welcome back!</h2> : <h2>Please log in.</h2>}


      {/* Button toggles the login state */}

      <button onClick={() => setIsLoggedIn(!isLoggedIn)}>

        {isLoggedIn ? "Log Out" : "Log In"}

      </button>

    </div>

  );

}
```


localhost:5173

Please log in.

OUTPUT:


localhost:5173

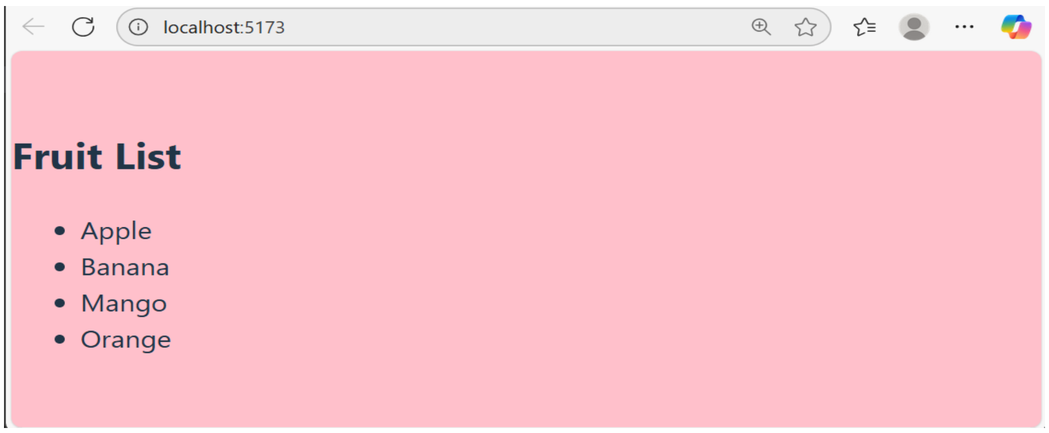
Welcome back!

b. Write a program for rendering lists.

App.jsx

```
import React from "react";
export default function App() {
  // A sample array of items
  const fruits = ["Apple", "Banana", "Mango", "Orange"];
  return (
    <div>
      <h2>Fruit List</h2>
      {/* Use map() to turn each item into an <li> */}
      <ul>
        {fruits.map((fruit, index) => (
          <li key={index}>{fruit}</li>
        ))}
      </ul>
    </div>
  );
}
```

OUTPUT:



localhost:5173

```
Fruit List

  • Apple
  • Banana
  • Mango
  • Orange
```

c. Write a program for working with different form fields using react forms

App.jsx

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

import "./App.css"; // import CSS file

export default function App() {

  const [formData, setFormData] = useState({

    username: "",

    email: "",

    role: "User",

    subscribe: false

  });


  const handleChange = (e) => {

    const { name, value, type, checked } = e.target;

    setFormData({

      ...formData,

      [name]: type === "checkbox" ? checked : value

    });

  };


  const handleSubmit = (e) => {

    e.preventDefault();

    alert("Form submitted! Check console for data.");
```

```
   console.log(formData);
  };

  return (
   <div className="container">
    <h2>Simple React Form</h2>
    <form onSubmit={handleSubmit} className="form">
     <div className="form-group">
      <label>Username:</label>
      <input
       type="text"
       name="username"
       value={formData.username}
       onChange={handleChange}
       placeholder="Enter username"
      />
     </div>

     <div className="form-group">
      <label>Email:</label>
      <input
       type="email"
       name="email"
       value={formData.email}
       onChange={handleChange}
      placeholder="Enter email"
    />
  </div>

  <div className="form-group">
    <label>Role:</label>
    <select name="role" value={formData.role} onChange={handleChange}>
      <option value="User">User</option>
      <option value="Admin">Admin</option>
      <option value="Moderator">Moderator</option>
    </select>
  </div>

  <div className="form-group checkbox">
    <label>
      <input
        type="checkbox"
        name="subscribe"
        checked={formData.subscribe}
        onChange={handleChange}
      /> Subscribe to newsletter
    </label>
  </div>

  <button type="submit" className="submit-btn">Submit</button>
```

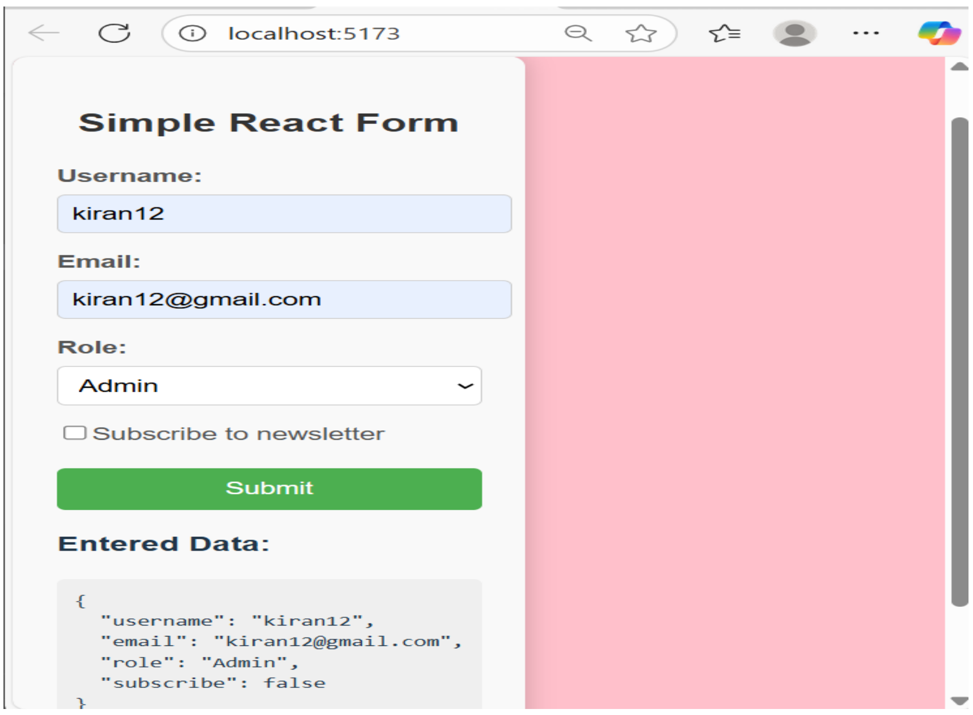
```
          </form>


          <h3>Entered Data:</h3>
          <pre className="output">{JSON.stringify(formData, null, 2)}</pre>
        </div>
      );
    }
```

App.css

```
/* Container styling */
.container {
  max-width: 500px;
  margin: 50px auto;
  padding: 25px;
  border-radius: 10px;
  box-shadow: 0px 0px 15px rgba(0,0,0,0.2);
  background-color: #f9f9f9;
  font-family: Arial, sans-serif;
}
/* Heading */
h2 {
  text-align: center;
  margin-bottom: 20px;
  color: #333;
}
/* Form layout */
.form {
  display: flex;
  flex-direction: column;
}
```

```
/* Form group spacing */
.form-group {
  margin-bottom: 15px;
}
/* Labels */
label {
  display: block;
  margin-bottom: 5px;
  font-weight: bold;
  color: #555;
}
/* Input and select fields */
input[type=”text”],
input[type=”email”],
select {
  width: 100%;
  padding: 8px;
  border-radius: 5px;
  border: 1px solid #ccc;
  font-size: 16px;
}

/* Checkbox spacing */
.checkbox label {
  font-weight: normal;
}
/* Submit button */
.submit-btn {
  padding: 10px;
  border-radius: 5px;
  border: none;
  background-color: #4caf50;
  color: white;
  font-size: 16px;
  cursor: pointer;
}
/* Submit button hover effect */
.submit-btn:hover {
  background-color: #45a049;
}
/* Output styling */
.output {
  background-color: #efefef;
  padding: 10px;
  border-radius: 5px;
  white-space: pre-wrap;
  word-wrap: break-word;
}
```

OUTPUT:



Simple React Form

Username:

kiran12

Email:

kiran12@gmail.com

Role:

Admin

☐ Subscribe to newsletter

Submit

Entered Data:

```
{
  "username": "kiran12",
  "email": "kiran12@gmail.com",
  "role": "Admin",
  "subscribe": false
}
```

8. ReactJS – React Router, Updating the Screen

a. Write a program for routing to different pages using react router.

```
npm install react-router-dom
```

App.jsx

```
import React from "react";
import { BrowserRouter as Router, Routes, Route, Link } from "react-router-dom";
import Home from "./Home";
import About from "./About";
import Contact from "./Contact";
import "./App.css"; // import CSS file

export default function App() {
  return (
    <Router>
      <div className="container">
        <h2>React Router Example</h2>

        {/* Navigation */}
        <nav className="nav">
          <Link to="/" className="nav-link">Home</Link>
          <Link to="/about" className="nav-link">About</Link>
          <Link to="/contact" className="nav-link">Contact</Link>
        </nav>

        {/* Routes */}
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="/contact" element={<Contact />} />
        </Routes>
      </div>
    </Router>
  );
}
```

Home.jsx

```
import React from "react";
export default function Home() {
  return <h3>Welcome to the Home Page!</h3>;
}
```

About.jsx

```
import React from "react";
export default function About() {
  return <h3>This is the About Page.</h3>;
}
```

Contact.jsx

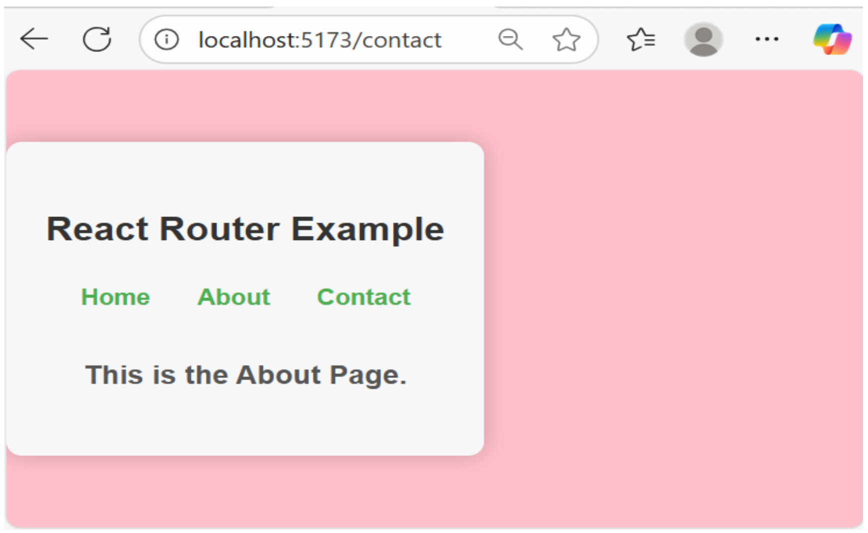
```
import React from "react";


export default function Contact() {
  return <h3>Get in touch on the Contact Page.</h3>;
}
```

App.css

```
/* Container styling */
.container {
  max-width: 600px;
  margin: 50px auto;
  padding: 25px;
  border-radius: 10px;
  box-shadow: 0px 0px 15px rgba(0,0,0,0.2);
  background-color: #f7f7f7;
  font-family: Arial, sans-serif;
  text-align: center;
}
/* Heading */
h2 {
  margin-bottom: 20px;
  color: #333;
}
/* Navigation styling */
.nav {
  margin-bottom: 30px;
}
```

```
.nav-link {
  margin: 0 15px;
  text-decoration: none;
  color: #4caf50;
  font-weight: bold;
  font-size: 16px;
  transition: color 0.3s;
}

.nav-link:hover {
  color: #388e3c;
}

/* Page content */
h3 {
  color: #555;
}
```

OUTPUT:

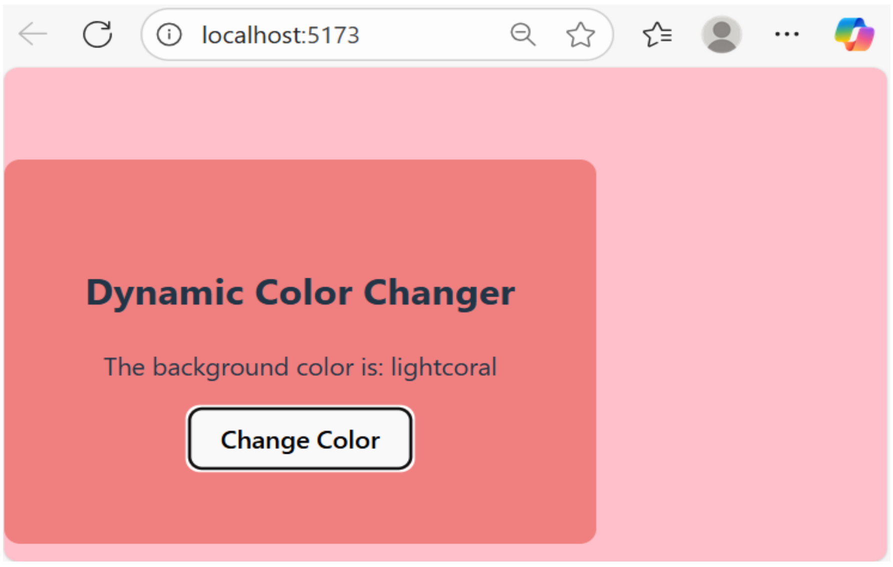


```
localhost:5173/contact

React Router Example

Home    About    Contact

This is the About Page.
```

b. Write a program for updating the screen.

App.jsx
```
import React, { useState } from "react";

export default function App() {
  const [color, setColor] = useState("lightblue");

  const changeColor = () => {
    const colors = ["lightblue", "lightgreen", "lightpink", "lightyellow", "lightcoral"];
    const randomColor = colors[Math.floor(Math.random() * colors.length)];
    setColor(randomColor); // Update state triggers screen update
  };

  return (
    <div
      style={{
        textAlign: "center",
        marginTop: "50px",
        padding: "50px",
        backgroundColor: color,
        borderRadius: "10px",
        transition: "background-color 0.5s",
      }}
    >
      <h2>Dynamic Color Changer</h2>
      <p>The background color is: {color}</p>
      <button onClick={changeColor}>Change Color</button>
    </div>
  );
}
```

OUTPUT:



Dynamic Color Changer

The background color is: lightcoral

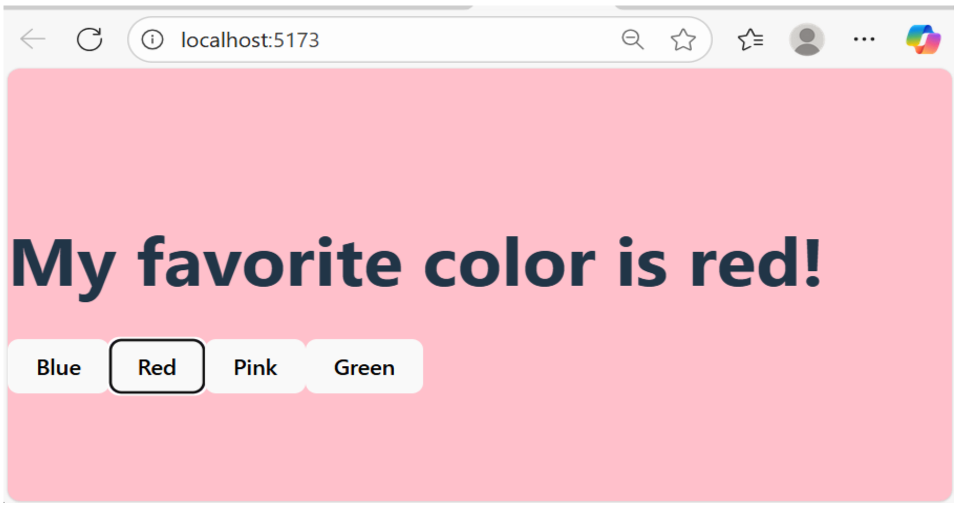
9. ReactJS – Hooks, Sharing data between Components

a.Write a program to understand the importance of using hooks.

App.jsx

```
mport { useState } from 'react';
import { createRoot } from 'react-dom/client';

export default function App() {
  const [color, setColor] = useState("red");

  return (
    <>
      <h1>My favorite color is {color}!</h1>
      <button
        type="button"
        onClick={() => setColor("blue")}
      >Blue</button>
      <button
        type="button"
        onClick={() => setColor("red")}
      >Red</button>
      <button
        type="button"
        onClick={() => setColor("pink")}
      >Pink</button>
      <button
        type="button"
        onClick={() => setColor("green")}
      >Green</button>
    </>
  );
}
```

OUTPUT:



localhost:5173

My favorite color is red!

b. Write a program for sharing data between components.

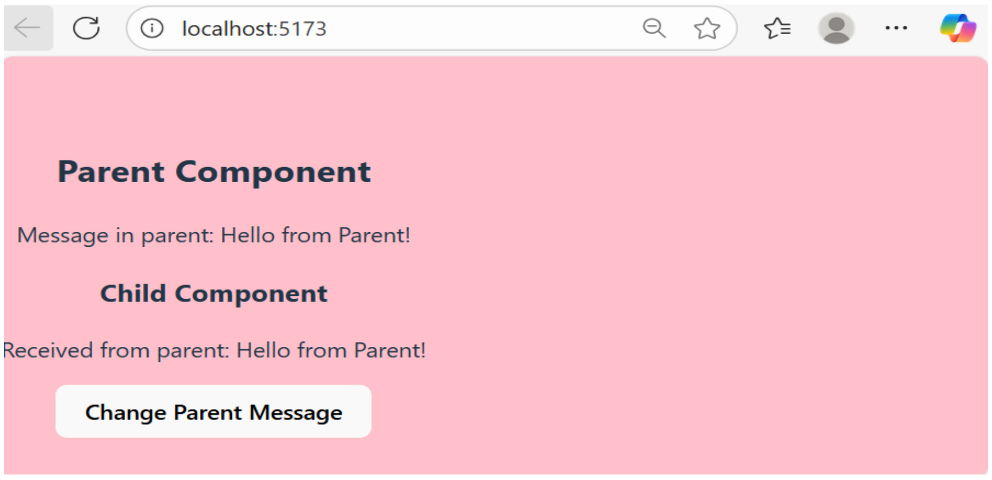
Child.jsx

```
import React from "react";

export default function Child({ message, updateMessage }) {
  return (
    <div style={{ marginTop: "20px" }}>
      <h3>Child Component</h3>
      <p>Received from parent: {message}</p>

      <button
        onClick={() => updateMessage("Message changed by Child!")}
      >
        Change Parent Message
      </button>
    </div>
  );
}
```

App.jsx

```
import React, { useState } from "react";
import Child from "./Child";

export default function App() {
  const [message, setMessage] = useState("Hello from Parent!");

  return (
    <div style={{ textAlign: "center", marginTop: "40px" }}>
      <h2>Parent Component</h2>
      <p>Message in parent: {message}</p>

      {/* Pass message and setMessage to Child as props */}
      <Child message={message} updateMessage={setMessage} />
    </div>
  );
}
```

OUTPUT:



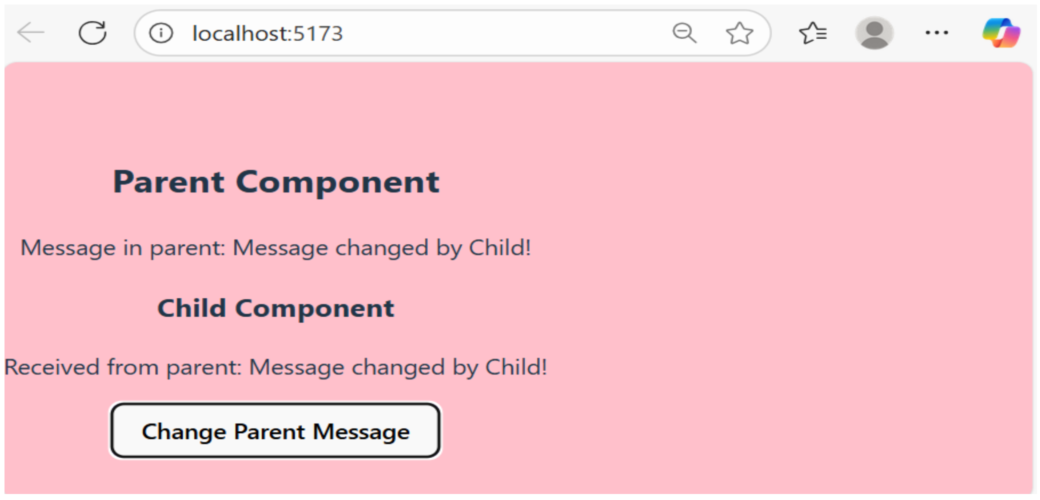
localhost:5173

Parent Component

Message in parent: Hello from Parent!

Child Component

Received from parent: Hello from Parent!



localhost:5173

Parent Component

Message in parent: Message changed by Child!

Child Component

Received from parent: Message changed by Child!