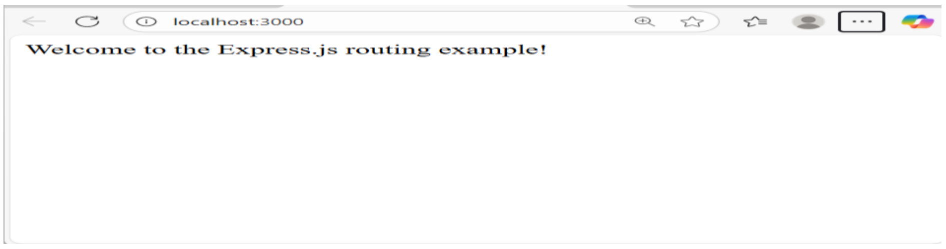
1. Express JS – Routing, HTTP Methods, Middleware.

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}`);
});
```

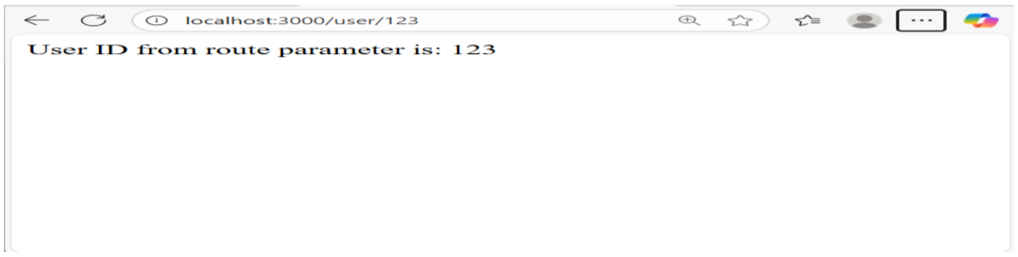
OUTPUT:

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



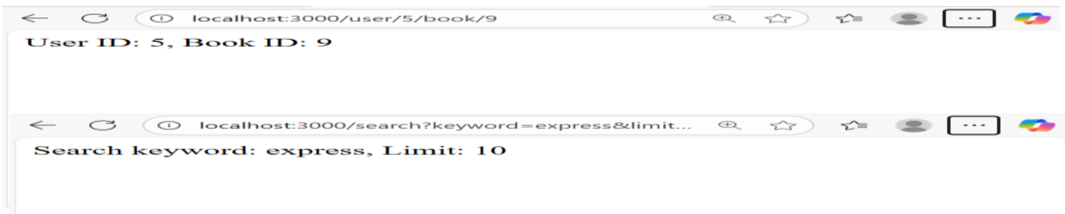
localhost:3000

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



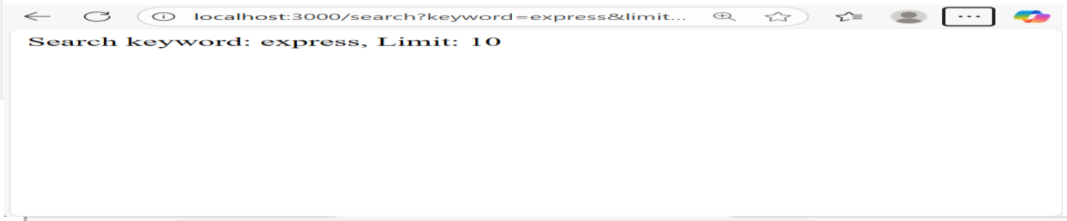
localhost:3000/user/123

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



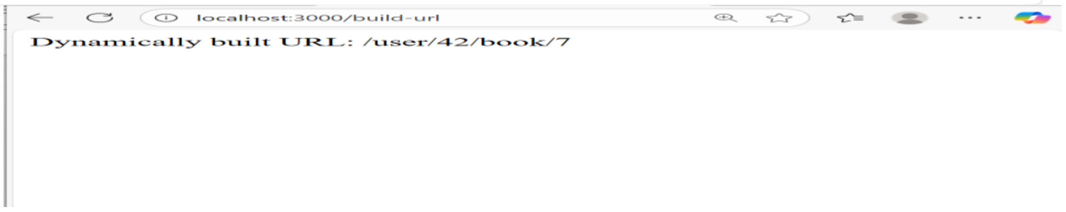
localhost:3000/user/5/book/9

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



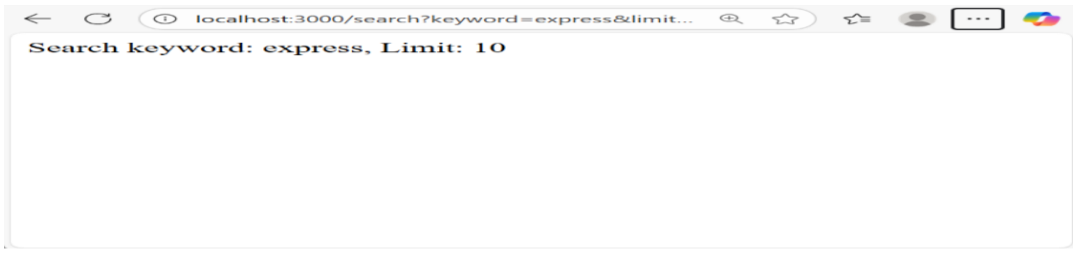
localhost:3000/search?keyword=express&limit...

```
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
```

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

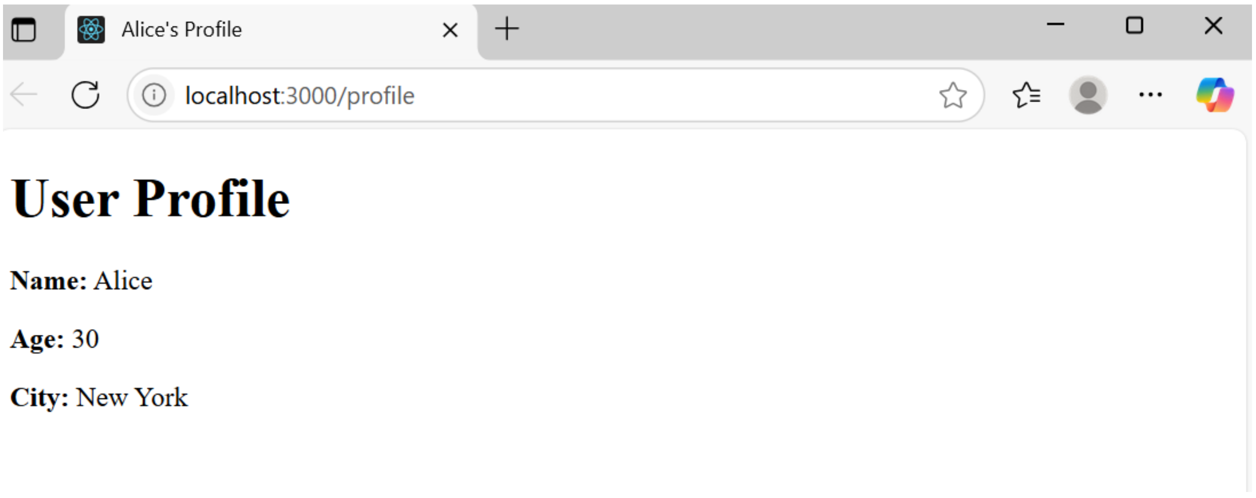
```
body {
   font-family: Arial, sans-serif;
   margin: 40px;
}
h1 {
   color: #2c3e50;
}
```

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

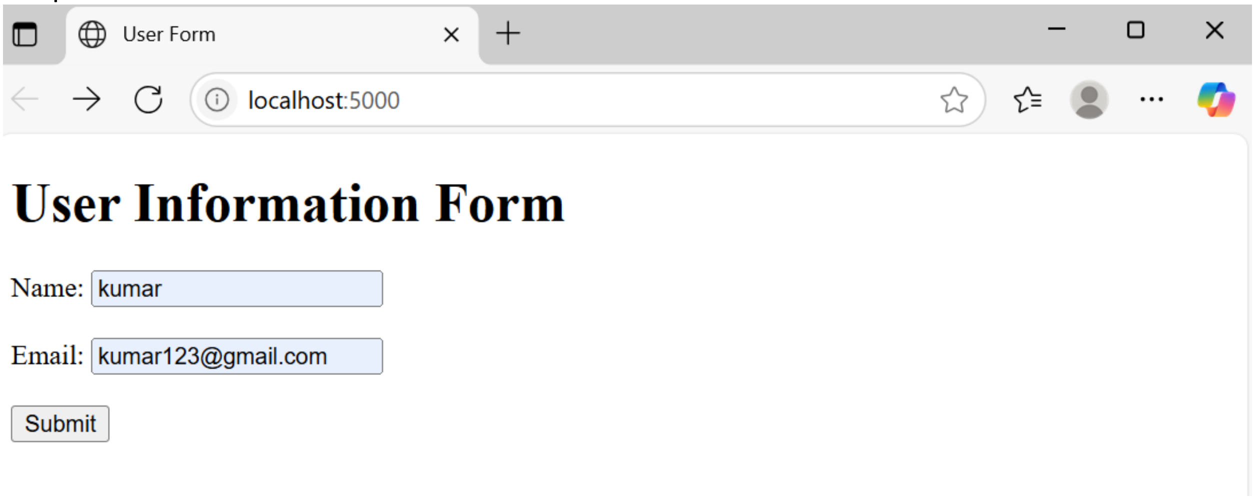
```
        const express = require('express');
        const app = express();
        const port = 5000;

        // 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:

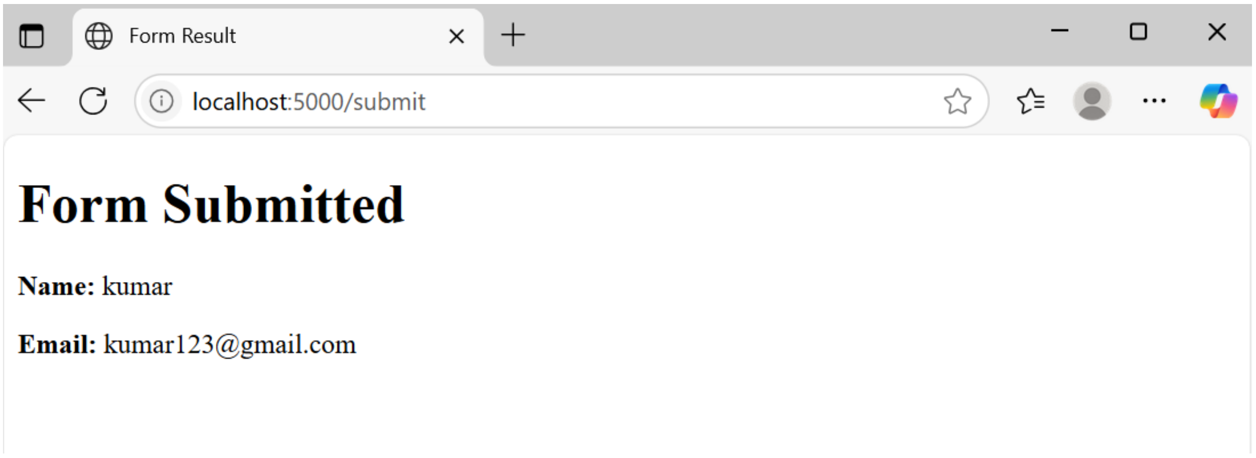


User Information Form

Name: kumar

Email: kumar123@gmail.com

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
```

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("/");
});
```

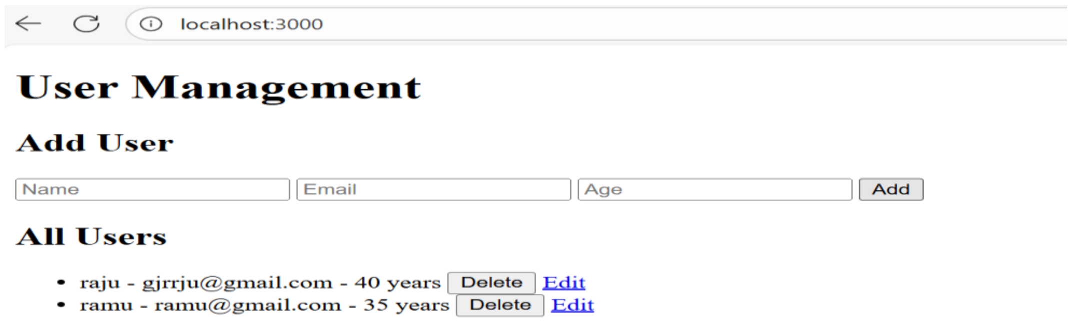
views/index.ejs

```
<!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

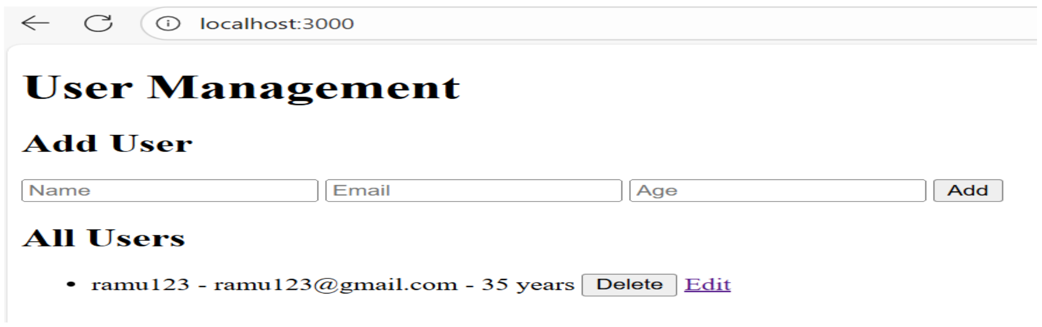
```
<!DOCTYPE html>
<html>
<head>
  <title>Edit User</title></head>
<body>
  <h1>Edit User</h1>
  <form action="/users/update/<%= user._id %>" method="POST">
    <input type="text" name="name" value="<%= user.name %>" required />
    <input type="email" name="email" value="<%= user.email %>" required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

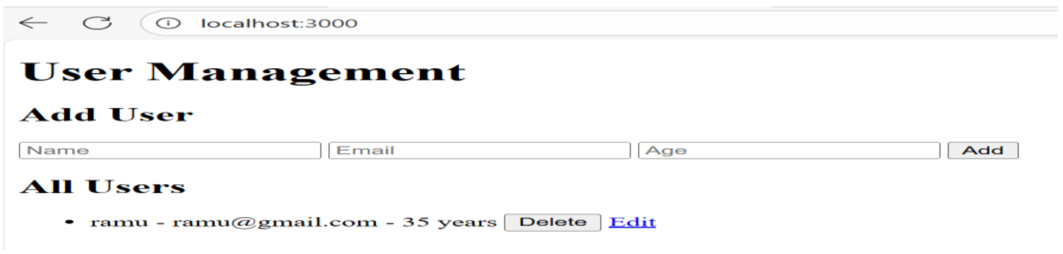
Add user



```
localhost:3000
User Management
Add User
Name  Email  Age  Add
All Users
• ramu123 - ramu123@gmail.com - 35 years Delete Edit
```

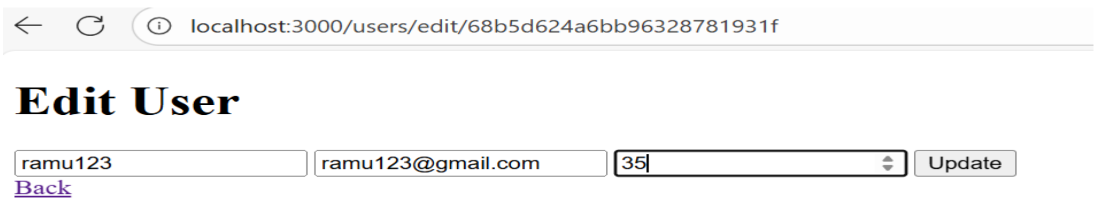
Delete

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
```

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

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

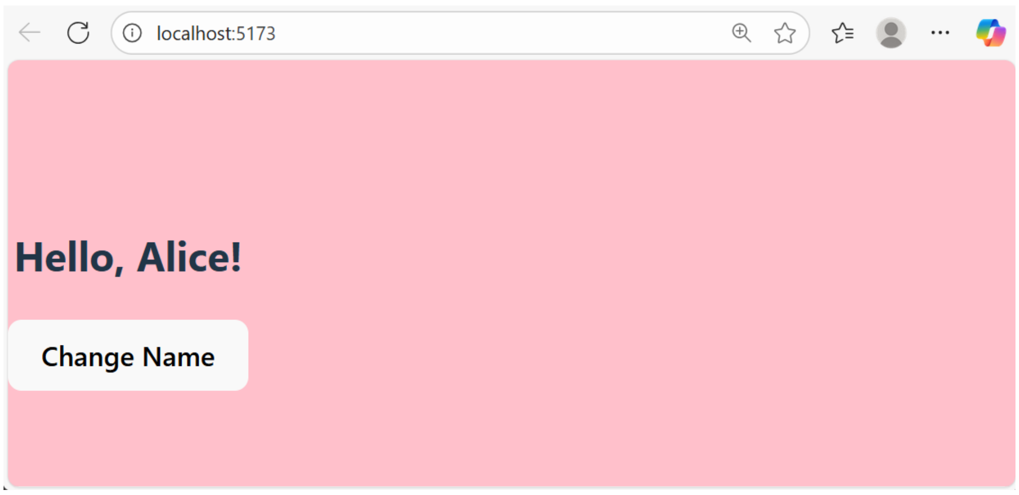
src/Greeting.jsx

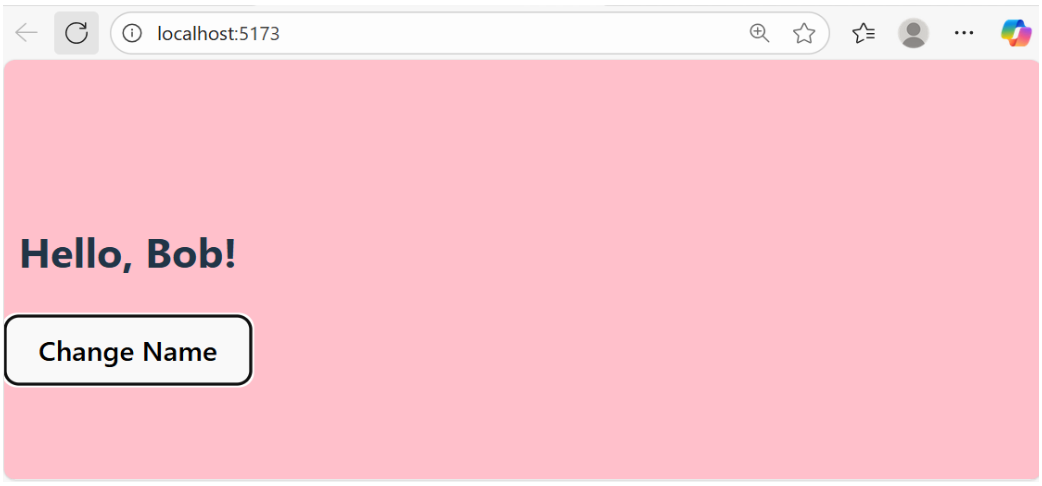
```
import React from "react";

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!
Change Name


localhost:5173
Hello, Bob!
Change Name

c) Program Responding to Events

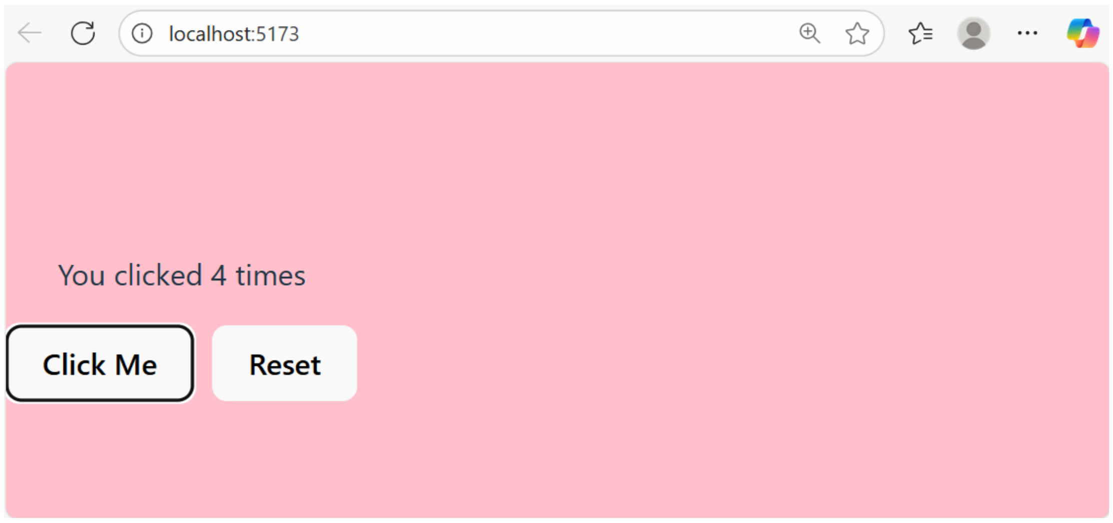
src/ClickCounter.jsx

```
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:



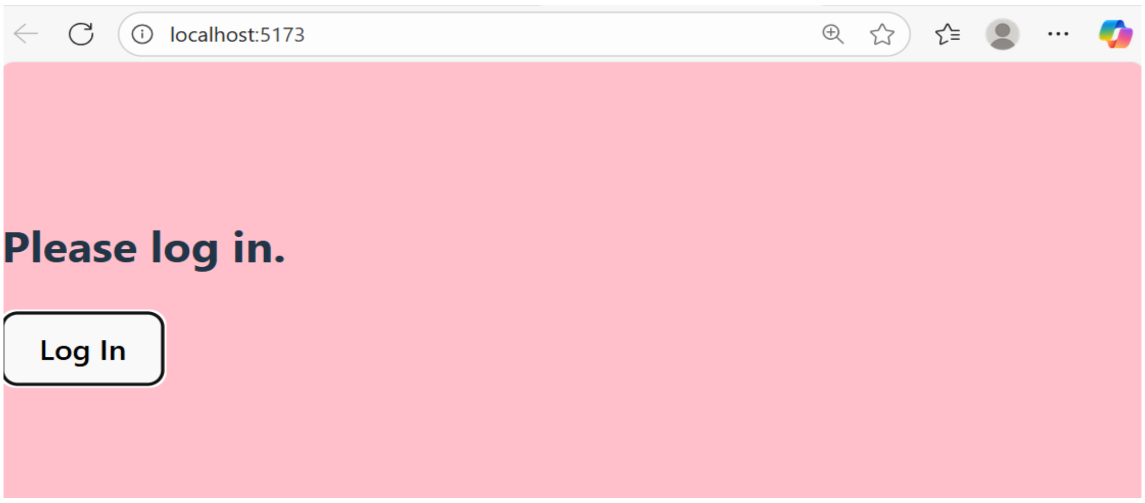
You clicked 4 times

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

a. Write a program for conditional rendering.

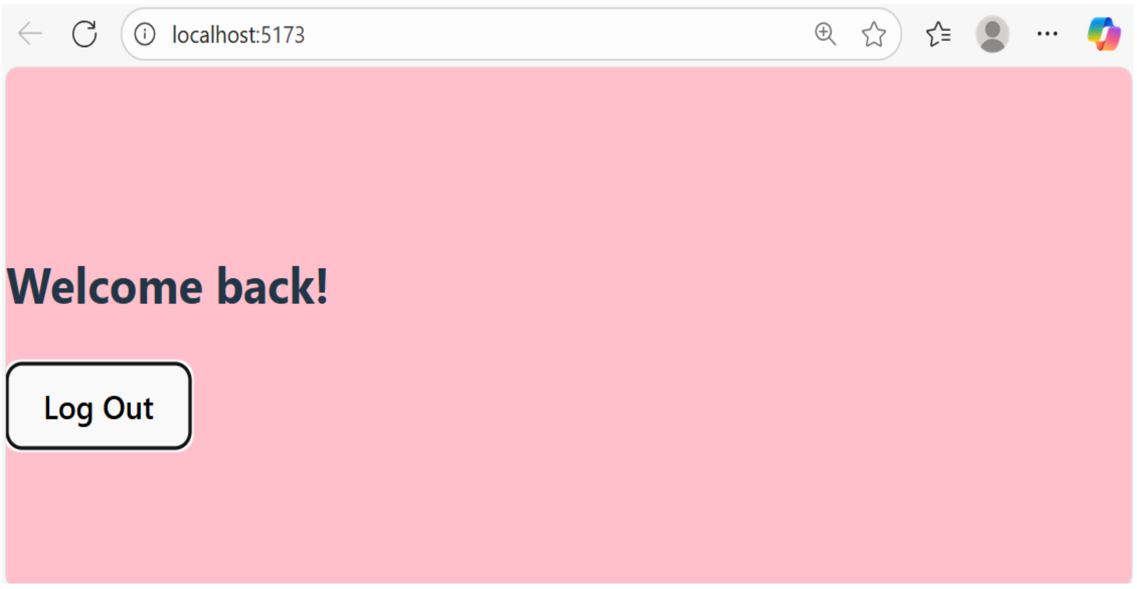
App.jsx

```
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

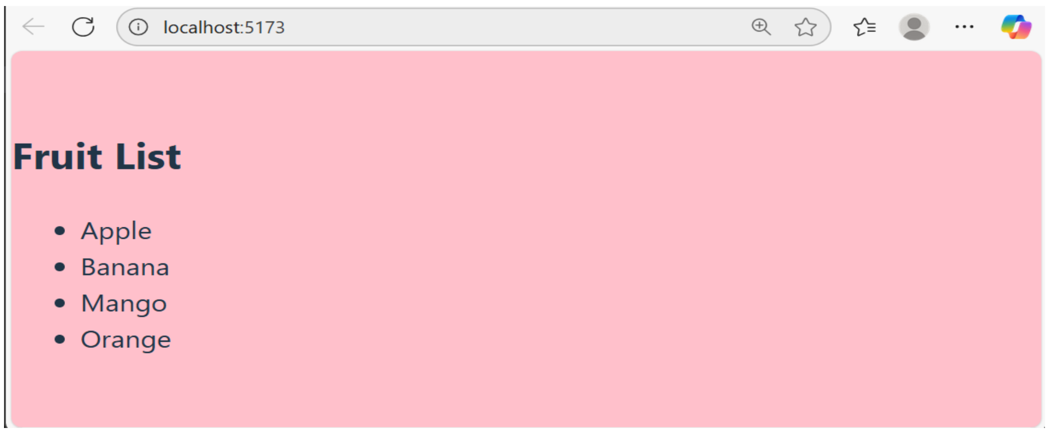
Welcome back!

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
```

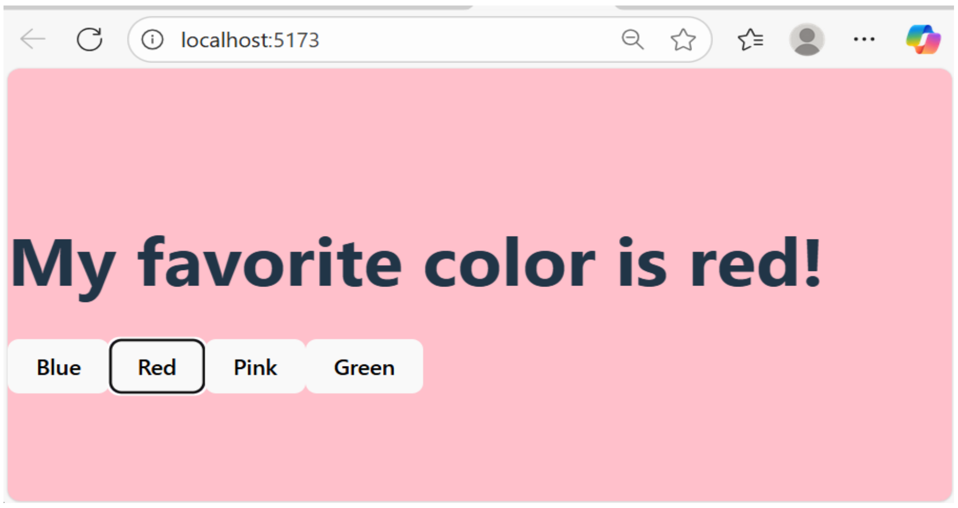
9. ReactJS – Hooks, Sharing data between Components

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!