Clean example of **login, logout, and signup** using **Express.js** with **MongoDB** (using Mongoose). This example will include **password hashing** and **basic session handling** using **express-session**.

Here's a step-by-step code snippet:

---

1. Setup Dependencies

```
mkdir auth-project
cd auth-project
npm init -y
npm install express mongoose bcryptjs express-session body-parser
```

We can refactor your Express app to follow a **modular structure**:

- **`server.js`** → main file to start the server
- **`models/User.js`** → Mongoose model
- **`routes/auth.js`** → all authentication routes (signup, login, logout)

Here's a clean example:

---

1. Project structure

```
project/
│
├─ server.js
├─ models/
│   └─ User.js
└─ routes/
    └─ auth.js
```

---

2. `models/User.js`

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

const userSchema = new mongoose.Schema({
    username: { type: String, required: true, unique: true },
```

```
    password: { type: String, required: true }
});

module.exports = mongoose.model('User', userSchema);
```

---

3. routes/auth.js

```
const express = require('express');
const bcrypt = require('bcryptjs');
const User = require('../models/User');

const router = express.Router();

// Signup
router.post('/signup', async (req, res) => {
    const { username, password } = req.body;
    if (!username || !password) return res.status(400).send('Please provide
username and password');

    const existingUser = await User.findOne({ username });
    if (existingUser) return res.status(400).send('User already exists');

    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = new User({ username, password: hashedPassword });
    await newUser.save();

    res.send('User registered successfully');
});

// Login
router.post('/login', async (req, res) => {
    const { username, password } = req.body;

    const user = await User.findOne({ username });
    if (!user) return res.status(400).send('Invalid username or password');

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).send('Invalid username or
password');

    req.session.userId = user._id; // save session
    res.send('Logged in successfully');
});

// Logout
router.post('/logout', (req, res) => {
    req.session.destroy(err => {
        if (err) return res.status(500).send('Could not log out.');
        res.send('Logged out successfully');
```

```
    });
});

module.exports = router;
```

---

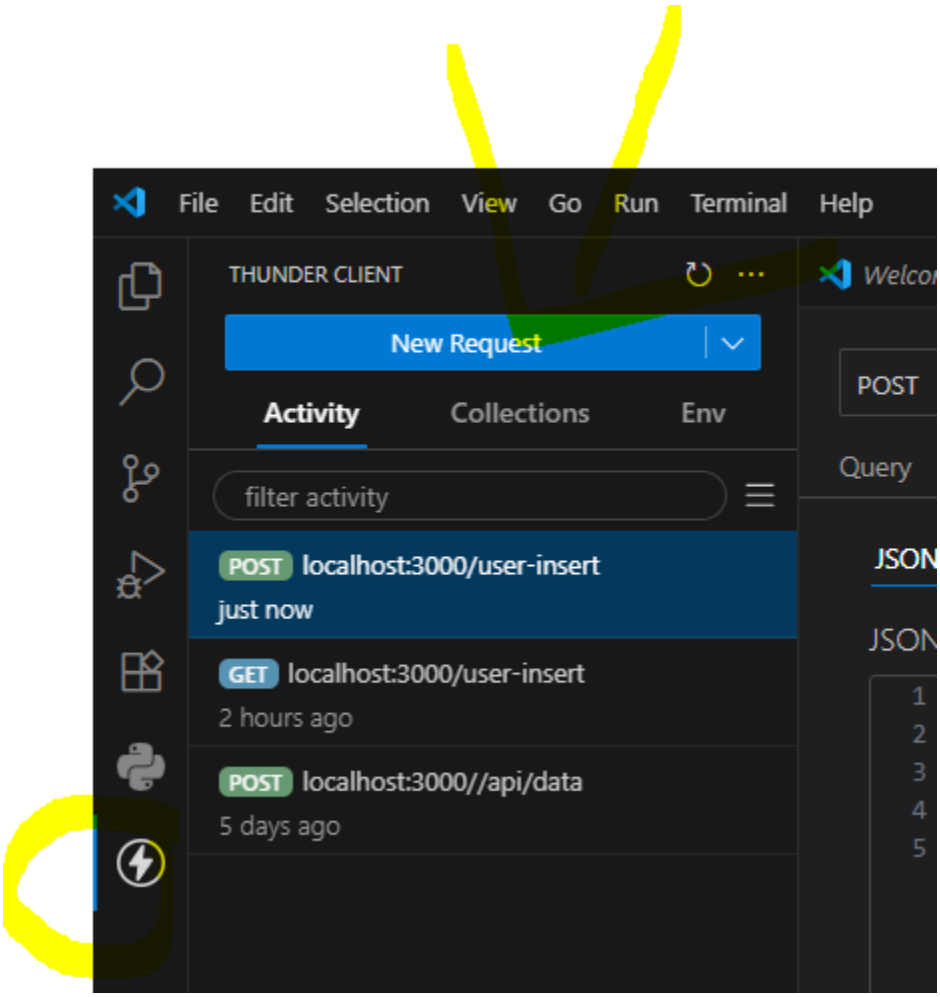
4. server.js

```
const express = require('express');
const mongoose = require('mongoose');
const session = require('express-session');
const bodyParser = require('body-parser');
const authRoutes = require('./routes/auth');

const app = express();

// Middleware
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

app.use(session({
    secret: 'kfj39f#23kfKJf!239kfjsdK!39dkf', // strong secret in production
    resave: false,
    saveUninitialized: false
}));

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

// Routes
app.use('/auth', authRoutes);

// Optional protected route
app.get('/dashboard', (req, res) => {
    if (!req.session.userId) return res.status(401).send('You must log in
first.');
    res.send('Welcome to your dashboard!');
});

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

 How it works:

- Signup: `POST /auth/signup`
- Login: `POST /auth/login`
- Logout: `POST /auth/logout`
- Protected: `GET /dashboard`

Let me show you **exactly how to test each route in ,Thunderclient in vs code extension as shown below** , including **sending JSON data and handling sessions**.



---

1. Signup: `POST /auth/signup`

URL:

```
http://localhost:3000/auth/signup
```

Method: POST

Headers:

```
Content-Type: application/json
```

Body (raw JSON):

```
{
    "username": "testuser",
    "password": "mypassword123"
}
```

Expected Response:

```
User registered successfully
```

---

2. Login: `POST /auth/login`

URL:

```
http://localhost:3000/auth/login
```

Method: POST

Headers:

```
Content-Type: application/json
```

Body (raw JSON):

```
{
```

```
    "username": "testuser",
    "password": "mypassword123"
}
```

Important:

- Make sure **"Enable Cookies"** in Postman is ON.
- After login, Postman will store the **session cookie**, which is required for logout and protected routes.

Expected Response:

```
Logged in successfully
```

---

3. Logout: `POST /auth/logout`

URL:

```
http://localhost:3000/auth/logout
```

Method: POST

Headers:

```
Content-Type: application/json
```

Body: None needed

Important:

- The session cookie from login will be sent automatically by Postman if cookies are enabled.

Expected Response:

```
Logged out successfully
```

---

4. Protected route: `GET /dashboard`

URL:

```
http://localhost:3000/dashboard
```

Method: GET

Headers:

```
Content-Type: application/json
```

Important:

- You must **login first** to have a valid session cookie.
- Postman will send the session cookie automatically if cookies are enabled.

Response if logged in:

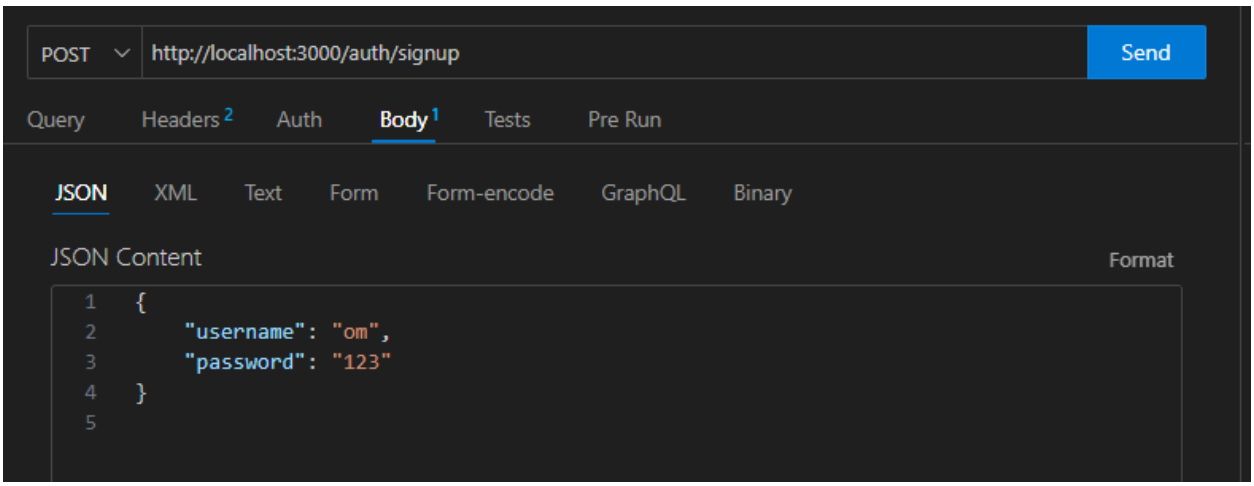
```
Welcome to your dashboard!
```

Response if not logged in:

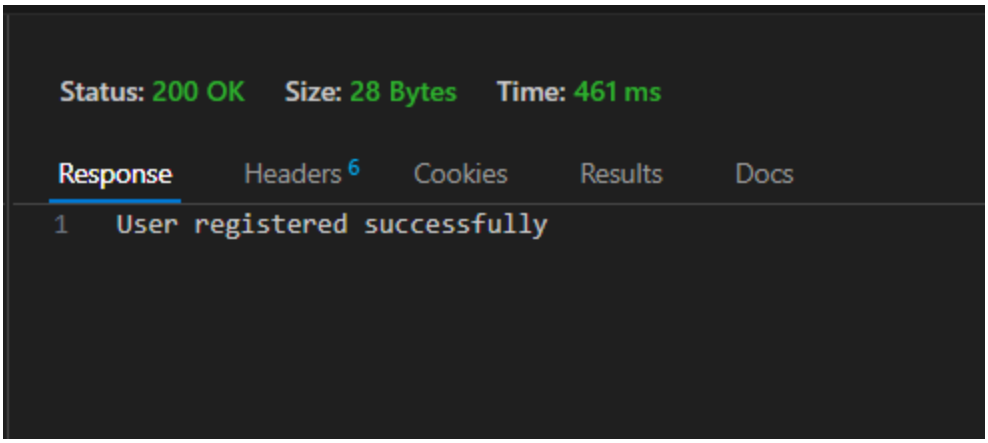
```
You must log in first.
```

See below examples how to test in thunder client :-

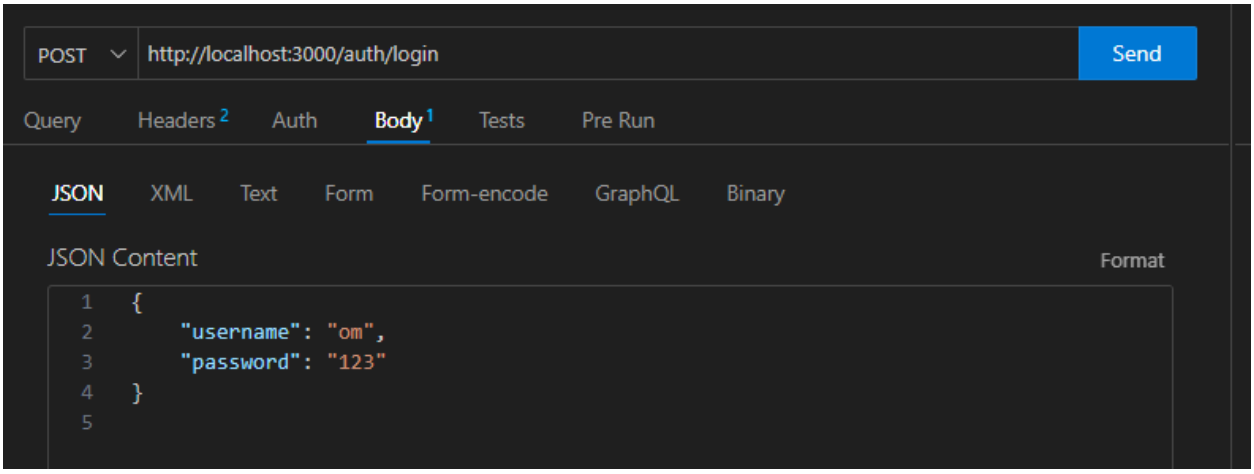
First signup :-



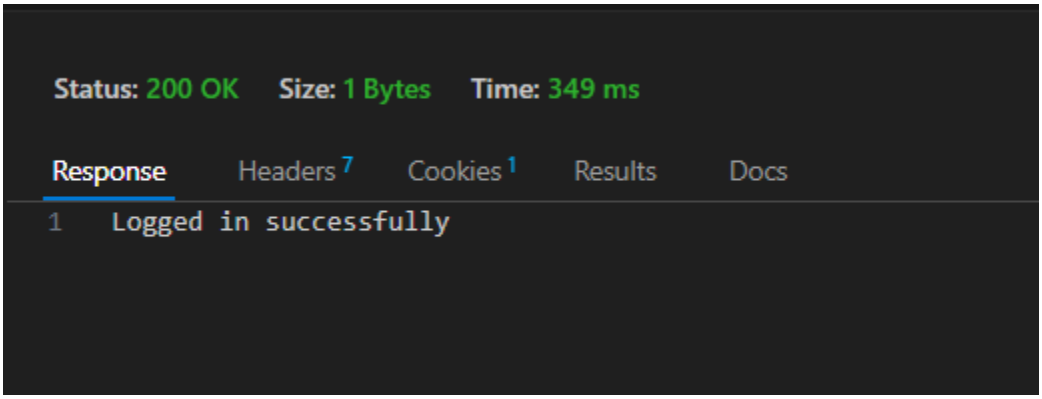
Response :-



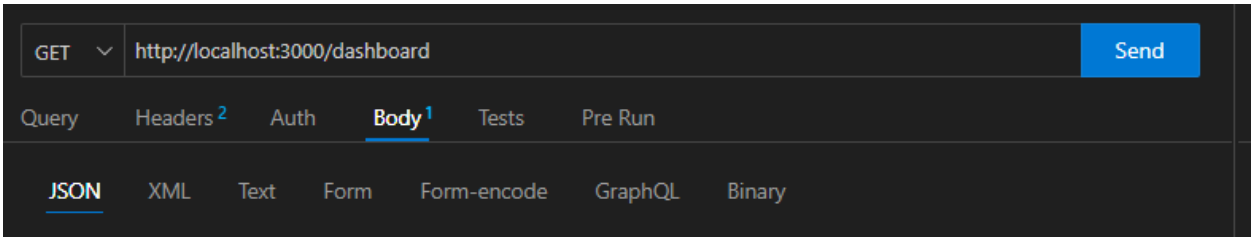
After it make login :-



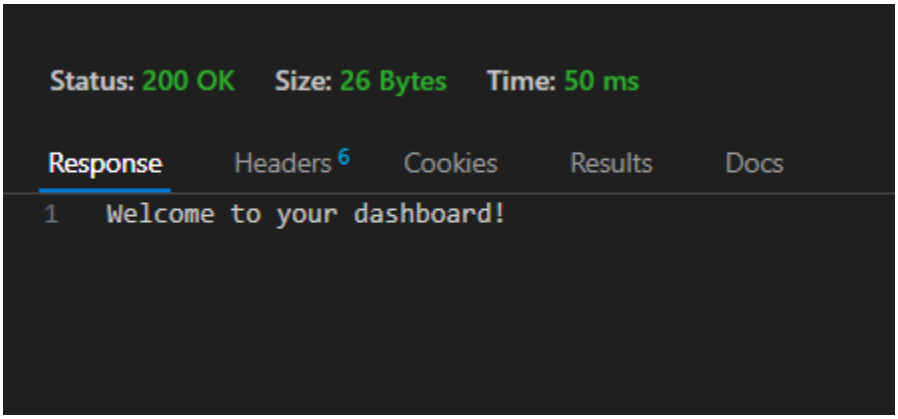
Response :-



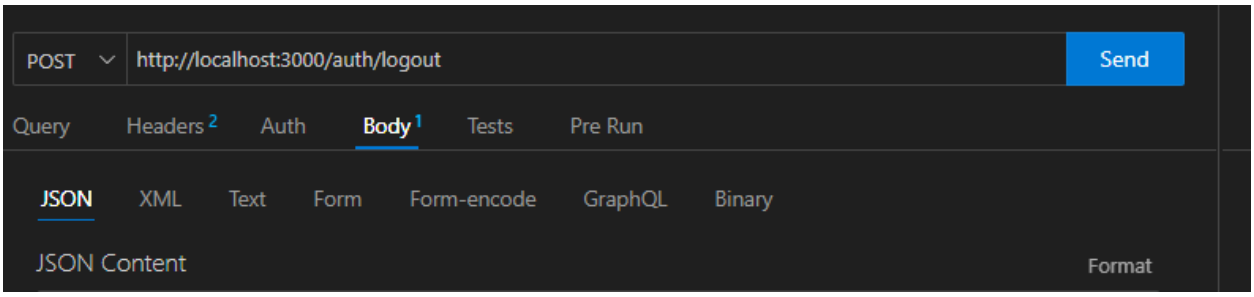
Get protected Data Dashboard:-



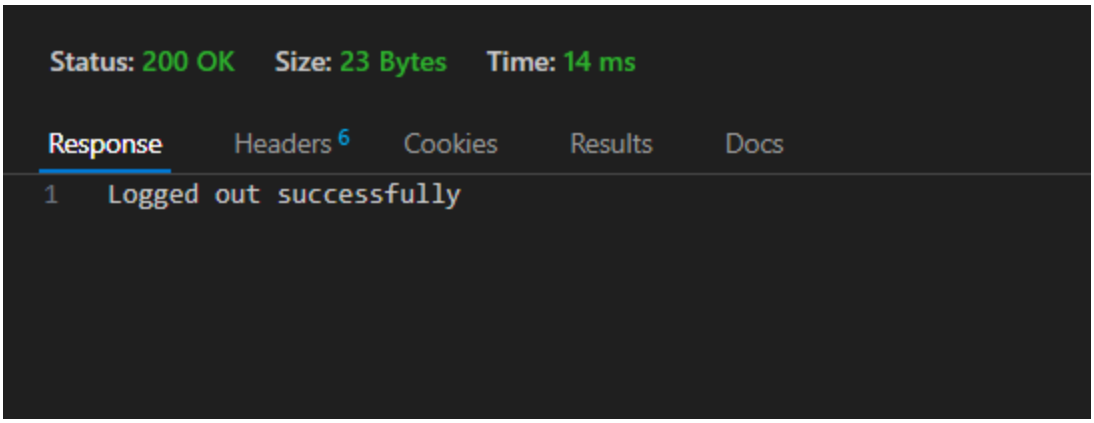
And response:-



And logout:-



And response:-


Status: 200 OK
Size: 23 Bytes
Time: 14 ms
Response
Headers 6
Cookies
Results
Docs
1
Logged out successfully

Let’s create a **simple React frontend** for your Express + MongoDB authentication system. This will include **Signup, Login, Logout, and Dashboard pages**. We'll use **Axios** for API requests and **React Router** for navigation.

Here’s a clean setup:

---

1. Create React App

```
npx create-react-app auth-frontend
cd auth-frontend
npm install axios react-router-dom
```

---

2. Project Structure

```
auth-frontend/
│
├─ src/
│   ├─ App.js
│   ├─ index.js
│   ├─ components/
│   │   ├─ Signup.js
│   │   ├─ Login.js
│   │   ├─ Dashboard.js
│   │   └─ Navbar.js
```

---

3. src/App.js

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Signup from './components/Signup';
import Login from './components/Login';
import Dashboard from './components/Dashboard';
import Navbar from './components/Navbar';

function App() {
  return (
    <Router>
      <Navbar />
      <div style={{ padding: '20px' }}>
        <Routes>
          <Route path="/signup" element={<Signup />} />
          <Route path="/login" element={<Login />} />
          <Route path="/dashboard" element={<Dashboard />} />
        </Routes>
      </div>
    </Router>
  );
}

export default App;
```

---

4. src/components/Navbar.js

```
import React from 'react';
import { Link } from 'react-router-dom';
import axios from 'axios';

const Navbar = () => {
  const handleLogout = async () => {
    try {
      await axios.post('http://localhost:3000/auth/logout', {}, {
withCredentials: true });
      alert('Logged out successfully');
    } catch (err) {
      console.log(err);
    }
  };

  return (
    <nav style={{ padding: '10px', borderBottom: '1px solid #ccc' }}>
      <Link to="/signup" style={{ marginRight: 10 }}>Signup</Link>
      <Link to="/login" style={{ marginRight: 10 }}>Login</Link>
      <Link to="/dashboard" style={{ marginRight: 10 }}>Dashboard</Link>
      <button onClick={handleLogout}>Logout</button>
    </nav>
  );
};

export default Navbar;
```

5. src/components/Signup.js

```
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

const Signup = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      await axios.post('http://localhost:3000/auth/signup', { username,
password });
      alert('User registered successfully');
      navigate('/login');
    } catch (err) {
      alert(err.response.data);
    }
  };

  return (
    <div>
      <h2>Signup</h2>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          placeholder="Username"
          value={username}
          onChange={e => setUsername(e.target.value)}
        /><br /><br />
        <input
          type="password"
          placeholder="Password"
          value={password}
          onChange={e => setPassword(e.target.value)}
        /><br /><br />
        <button type="submit">Signup</button>
      </form>
    </div>
  );
};

export default Signup;
```

6. src/components/Login.js

```
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

const Login = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      await axios.post('http://localhost:3000/auth/login', { username,
password }, { withCredentials: true });
      alert('Logged in successfully');
      navigate('/dashboard');
    } catch (err) {
      alert(err.response.data);
    }
  };

  return (
    <div>
      <h2>Login</h2>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          placeholder="Username"
          value={username}
          onChange={e => setUsername(e.target.value)}
        /><br /><br />
        <input
          type="password"
          placeholder="Password"
          value={password}
          onChange={e => setPassword(e.target.value)}
        /><br /><br />
        <button type="submit">Login</button>
      </form>
    </div>
  );
};

export default Login;
```

---

7. src/components/Dashboard.js

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

const Dashboard = () => {
  const [message, setMessage] = useState('');

  useEffect(() => {
    const fetchData = async () => {
      try {
        const res = await axios.get('http://localhost:3000/dashboard', {
withCredentials: true });
        setMessage(res.data);
      } catch (err) {
        setMessage(err.response.data);
      }
    };
    fetchData();
  }, []);

  return (
    <div>
      <h2>Dashboard</h2>
      <p>{message}</p>
    </div>
  );
};

export default Dashboard;
```

---

8. Notes in your server.js file code:-

1. **CORS on backend**: Since React runs on a different port (3000 by default), you need to enable CORS in Express:

```
const cors = require('cors');
app.use(cors({ origin: 'http://localhost:3000', credentials: true }));
```

Install CORS:

```
npm install cors
```

2. **withCredentials**: Important for sending **cookies** (session-based auth).

After updating you will see code:-

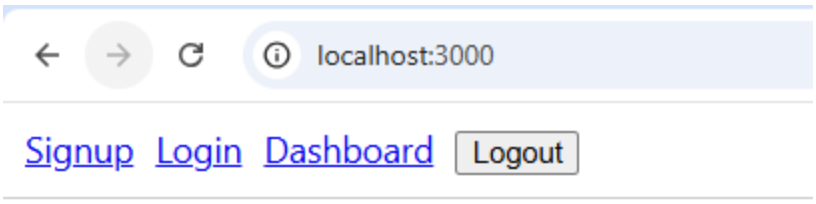
const express = require('express');

```
const mongoose = require('mongoose');
const session = require('express-session');
const bodyParser = require('body-parser');
const authRoutes = require('./routes/auth');

const app = express();

const cors = require('cors');
app.use(cors({ origin: 'http://localhost:3000', credentials: true }));

// Middleware
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

app.use(session({
    secret: 'kfj39f#23kfKJf!239kfjsdK!39dkf', // strong secret in production
    resave: false,
    saveUninitialized: false
}));

// Connect to MongoDB
mongoose.connect('mongodb://127.0.0.1:27017/authDemo', {
    useNewUrlParser: true,
```

```
    useUnifiedTopology: true
}).then(() => console.log('MongoDB connected'))
  .catch(err => console.log(err));

// Routes
app.use('/auth', authRoutes);

// Optional protected route
app.get('/dashboard', (req, res) => {
    if (!req.session.userId) return res.status(401).send('You must log in first.');
    res.send('Welcome to your dashboard!');
});

// Start server
app.listen(5000, () => {
    console.log('Server running on http://localhost:5000');
});
```

This frontend lets you:

- Signup → POST `/auth/signup`
- Login → POST `/auth/login`
- Logout → POST `/auth/logout`
- View protected Dashboard → GET `/dashboard`



Note:-

For production in your backend side create .env file and in serverjs file change following code:

Use your real frontend domain and environment variables:

```
const cors = require('cors');
const allowedOrigin = process.env.FRONTEND_URL || 'http://localhost:3000';

app.use(cors({
  origin: allowedOrigin,
  credentials: true,
}));
```

In your `.env` file:

```
FRONTEND_URL=https://your-frontend-domain.com
```

If you want to extend your existing **login/signup system** so that once an **admin logs in**, they can perform **CRUD operations** on a `Record` collection (fields: `name`, `email`, `city`).

Let's do this the **right way**, step by step

Overview

You already have:

- Express + MongoDB + Mongoose
- Session-based authentication

We'll now add:

- A **Record model**
- **CRUD routes** (`Create`, `Read`, `Update`, `Delete`)
- A **middleware** to protect these routes (only logged-in users can access)

Project Structure

```
project/
├─ server.js
├─ models/
│   ├─ User.js
│   └─ Record.js
└─ routes/
    ├─ auth.js
    └─ record.js
```

---

1 models/Record.js

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

const recordSchema = new mongoose.Schema({
    name: { type: String, required: true },
    email: { type: String, required: true },
    city: { type: String, required: true },
});

module.exports = mongoose.model('Record', recordSchema);
```

---

2 middleware/authMiddleware.js

Create a small middleware to protect routes (only logged-in users can access).

```
module.exports = (req, res, next) => {
    if (!req.session.userId) {
        return res.status(401).send('Unauthorized: Please log in first.');
    }
    next();
};
```

---

3 routes/record.js

```
const express = require('express');
const Record = require('../models/Record');
const authMiddleware = require('../middleware/authMiddleware');

const router = express.Router();

// Create record
router.post('/', authMiddleware, async (req, res) => {
    const { name, email, city } = req.body;
    try {
        const record = new Record({ name, email, city });
        await record.save();
        res.status(201).send('Record created successfully');
    } catch (err) {
        res.status(500).send(err.message);
    }
});

// Get all records
router.get('/', authMiddleware, async (req, res) => {
    try {
        const records = await Record.find();
        res.json(records);
    } catch (err) {
        res.status(500).send(err.message);
    }
});

// Update record
router.put('/:id', authMiddleware, async (req, res) => {
    const { name, email, city } = req.body;
    try {
        const updated = await Record.findByIdAndUpdate(
            req.params.id,
            { name, email, city },
            { new: true }
        );
```

```
        if (!updated) return res.status(404).send('Record not found');
        res.send('Record updated successfully');
    } catch (err) {
        res.status(500).send(err.message);
    }
});

// □ Delete record
router.delete('/:id', authMiddleware, async (req, res) => {
    try {
        const deleted = await Record.findByIdAndDelete(req.params.id);
        if (!deleted) return res.status(404).send('Record not found');
        res.send('Record deleted successfully');
    } catch (err) {
        res.status(500).send(err.message);
    }
});

module.exports = router;
```

4□ server.js

Now connect your new routes and middleware.

```
const express = require('express');
const mongoose = require('mongoose');
const session = require('express-session');
const bodyParser = require('body-parser');
const cors = require('cors');
const authRoutes = require('./routes/auth');
const recordRoutes = require('./routes/record');

const app = express();

// --- Middleware ---
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

app.use(cors({
    origin: 'http://localhost:3000',
    credentials: true
}));

app.use(session({
    secret: 'kfj39f#23kfKJf!239kfjsdK!39dkf',
    resave: false,
    saveUninitialized: false,
}));

// --- MongoDB Connection ---
mongoose.connect('mongodb://127.0.0.1:27017/authDemo', {
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(() => console.log('MongoDB connected'))
```

```
  .catch(err => console.log(err));

// --- Routes ---
app.use('/auth', authRoutes);
app.use('/records', recordRoutes);

app.get('/dashboard', (req, res) => {
    if (!req.session.userId) return res.status(401).send('You must log in
first.');
    res.send('Welcome to your dashboard!');
});

// --- Server Start ---
app.listen(5000, () => console.log('Server running on
http://localhost:5000'));
```

Postman Testing

You can now test these new endpoints after logging in:

Operation	Method	Endpoint	Body Example
Create	POST	`/records`	`{ "name": "John", "email": "john@gmail.com", "city": "NY" }`
Read All	GET	`/records`	*(none)*
Update	PUT	`/records/<id>`	`{ "name": "John Doe", "email": "john@gmail.com", "city": "LA" }`
Delete	DELETE	`/records/<id>`	*(none)*

And then in you frontend code make following changes :-

Let's extend your existing **React frontend** to include an **Admin Dashboard** where the logged-in admin can perform **CRUD operations** on the `Record` collection (`name`, `email`, `city`).

We'll keep it simple but fully functional — using **Axios** and **React Router**.

📁 Updated Folder Structure

```
auth-frontend/
│
├─ src/
│   ├─ App.js
│   ├─ index.js
│   ├─ components/
│   │   ├─ Navbar.js
│   │   ├─ Signup.js
│   │   ├─ Login.js
│   │   ├─ Dashboard.js   <-- admin CRUD
│   │   ├─ RecordForm.js  <-- add/edit form
│   │   └─ RecordList.js  <-- list of records
```

🧩 1. Install dependencies

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

📄 2. `src/App.js`

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Signup from './components/Signup';
import Login from './components/Login';
import Dashboard from './components/Dashboard';
import Navbar from './components/Navbar';

function App() {
  return (
    <Router>
      <Navbar />
      <div style={{ padding: '20px' }}>
        <Routes>
          <Route path="/signup" element={<Signup />} />
          <Route path="/login" element={<Login />} />
          <Route path="/dashboard" element={<Dashboard />} />
        </Routes>
      </div>
    </Router>
  );
}

export default App;
```

---

3. src/components/Navbar.js

```
import React from 'react';
import { Link } from 'react-router-dom';
import axios from 'axios';

const Navbar = () => {
  const handleLogout = async () => {
    try {
      await axios.post('http://localhost:3000/auth/logout', {}, {
withCredentials: true });
      alert('Logged out successfully');
      window.location.href = '/login';
    } catch (err) {
      console.log(err);
    }
  };

  return (
    <nav style={{ padding: '10px', borderBottom: '1px solid #ccc' }}>
      <Link to="/signup" style={{ marginRight: 10 }}>Signup</Link>
      <Link to="/login" style={{ marginRight: 10 }}>Login</Link>
      <Link to="/dashboard" style={{ marginRight: 10 }}>Dashboard</Link>
      <button onClick={handleLogout}>Logout</button>
    </nav>
  );
};

export default Navbar;
```

---

4. src/components/Dashboard.js

This is the **admin CRUD dashboard**.

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import RecordForm from './RecordForm';
import RecordList from './RecordList';

const Dashboard = () => {
  const [records, setRecords] = useState([]);
  const [editingRecord, setEditingRecord] = useState(null);

  // Fetch all records
  const fetchRecords = async () => {
    try {
      const res = await axios.get('http://localhost:3000/records', {
withCredentials: true });
      setRecords(res.data);
    } catch (err) {
      alert('Please login as admin to view this page');
    }
  };

  useEffect(() => {
```

```
    fetchRecords();
  }, []);

  // Create or update record
  const handleSave = async (record) => {
    try {
      if (editingRecord) {
        await axios.put(`http://localhost:5000/records/${editingRecord._id}`,
record, { withCredentials: true });
        alert('Record updated');
      } else {
        await axios.post('http://localhost:5000/records', record, {
withCredentials: true });
        alert('Record created');
      }
      setEditingRecord(null);
      fetchRecords();
    } catch (err) {
      alert(err.response?.data || 'Error saving record');
    }
  };

  // Delete record
  const handleDelete = async (id) => {
    if (!window.confirm('Delete this record?')) return;
    try {
      await axios.delete(`http://localhost:5000/records/${id}`, {
withCredentials: true });
      alert('Record deleted');
      fetchRecords();
    } catch (err) {
      alert(err.response?.data || 'Error deleting record');
    }
  };

  return (
    <div>
      <h2>Admin Dashboard</h2>
      <RecordForm onSave={handleSave} editingRecord={editingRecord} />
      <RecordList records={records} onEdit={setEditingRecord}
onDelete={handleDelete} />
    </div>
  );
};

export default Dashboard;
```

---

5. `src/components/RecordForm.js`

Form for **adding or editing** records.

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

const RecordForm = ({ onSave, editingRecord }) => {
  const [form, setForm] = useState({ name: '', email: '', city: '' });

  useEffect(() => {
    if (editingRecord) setForm(editingRecord);
    else setForm({ name: '', email: '', city: '' });
  }, [editingRecord]);

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

  const handleSubmit = (e) => {
    e.preventDefault();
    onSave(form);
  };

  return (
    <form onSubmit={handleSubmit} style={{ marginBottom: '20px' }}>
      <h3>{editingRecord ? 'Edit Record' : 'Add Record'}</h3>
      <input name="name" placeholder="Name" value={form.name}
onChange={handleChange} />{' '}
      <input name="email" placeholder="Email" value={form.email}
onChange={handleChange} />{' '}
      <input name="city" placeholder="City" value={form.city}
onChange={handleChange} />{' '}
      <button type="submit">{editingRecord ? 'Update' : 'Add'}</button>
    </form>
  );
};

export default RecordForm;
```

---

6. `src/components/RecordList.js`

Displays all records with edit/delete buttons.

```
import React from 'react';

const RecordList = ({ records, onEdit, onDelete }) => {
  if (records.length === 0) return <p>No records found.</p>;

  return (
    <table border="1" cellPadding="8">
      <thead>
        <tr>
          <th>Name</th>
          <th>Email</th>
          <th>City</th>
          <th>Actions</th>
        </tr>
      </thead>
      <tbody>
        {records.map((r) => (
          <tr key={r._id}>
            <td>{r.name}</td>
            <td>{r.email}</td>
            <td>{r.city}</td>
            <td>
              <button onClick={() => onEdit(r)}>Edit</button>{' '}
              <button onClick={() => onDelete(r._id)}>Delete</button>
            </td>
          </tr>
        ))}
      </tbody>
    </table>
  );
};

export default RecordList;
```

---

7. Update Backend CORS (if not done yet)

In your Express backend (`server.js`):

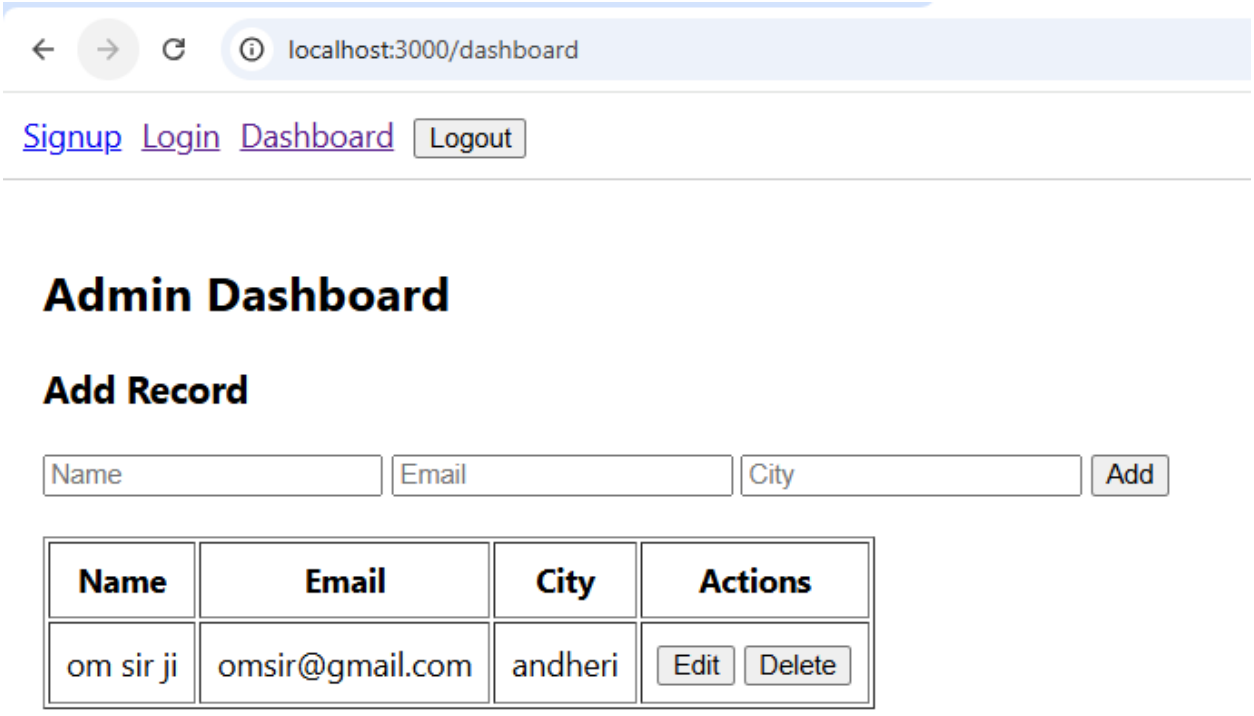
```
const cors = require('cors');
app.use(cors({
  origin: 'http://localhost:3000',
  credentials: true,
}));
```

---

8. How It Works

Feature	Request	Route	Description
View Records	`GET`	`/records`	Lists all records
Add Record	`POST`	`/records`	Add new record
Edit Record	`PUT`	`/records/:id`	Update record
Delete Record	`DELETE`	`/records/:id`	Delete record

- Only logged-in users (admins, ideally) can access `/records` routes.
- Session cookies are automatically sent (`withCredentials: true`).



Perfect — you want your **Navbar** to dynamically show either:-

- **"Login" and "Signup"** when the user is **logged out**,
- or **"Dashboard" and "Logout"** when the user is **logged in**

We can do this easily by checking the **user's session** using an API call to your backend (e.g., `/auth/check`), or by tracking login state locally in React.

Let's make it clean and reliable

---

1. Update your backend first (add `/auth/check` route)

In your `routes/auth.js`, add this simple route:

```
// Check if user is logged in
router.get('/check', (req, res) => {
    if (req.session.userId) {
        res.json({ loggedIn: true });
    } else {
        res.json({ loggedIn: false });
    }
});
```

 This will return `{ loggedIn: true }` if the user has an active session.

---

2. Updated Navbar.js

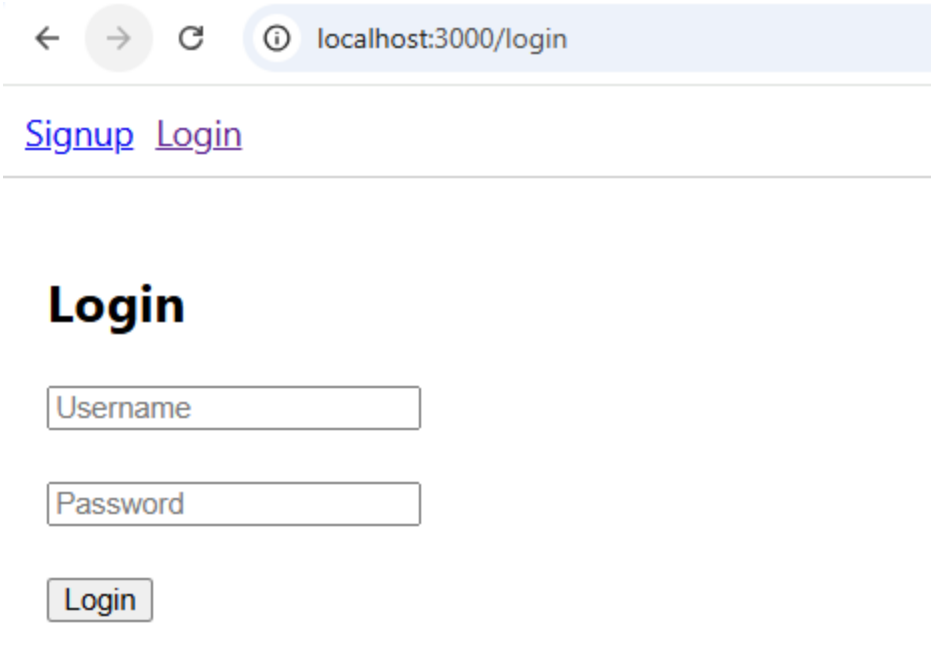
Here's the updated React component:

```
import React, { useEffect, useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import axios from 'axios';

const Navbar = () => {
  const [loggedIn, setLoggedIn] = useState(false);
  const navigate = useNavigate();

  // Check login status on mount
  useEffect(() => {
    const checkLogin = async () => {
      try {
```

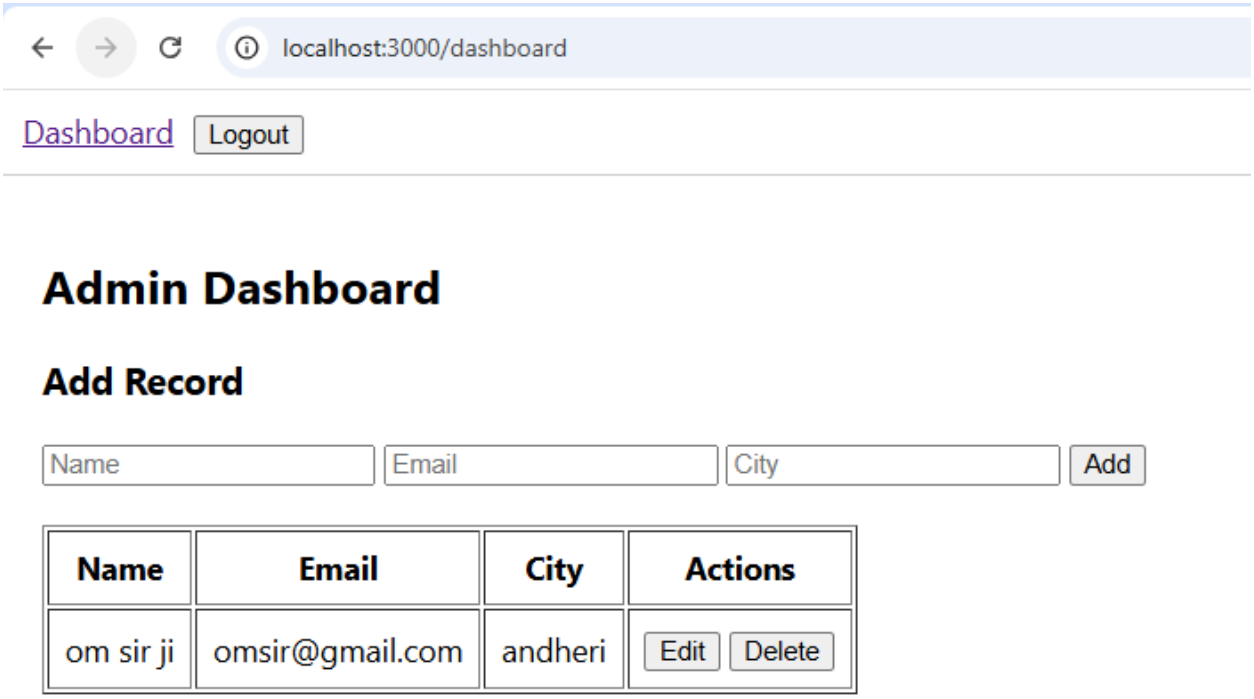
```
        const res = await axios.get('http://localhost:5000/auth/check', {
withCredentials: true });
        setLoggedIn(res.data.loggedIn);
      } catch (err) {
        console.error('Error checking login:', err);
      }
    };
    checkLogin();
  }, []);

  // Handle logout
  const handleLogout = async () => {
    try {
      await axios.post('http://localhost:5000/auth/logout', {}, {
withCredentials: true });
      setLoggedIn(false);
      alert('Logged out successfully');
      navigate('/login');
    } catch (err) {
      console.error(err);
    }
  };

  return (
    <nav style={{ padding: '10px', borderBottom: '1px solid #ccc' }}>
      {!loggedIn ? (
        <>
          <Link to="/signup" style={{ marginRight: 10 }}>Signup</Link>
          <Link to="/login" style={{ marginRight: 10 }}>Login</Link>
        </>
      ) : (
        <>
          <Link to="/dashboard" style={{ marginRight: 10 }}>Dashboard</Link>
          <button onClick={handleLogout}>Logout</button>
        </>
      )}
    </nav>
  );
};

export default Navbar;
```

How It Works

- When the Navbar loads, it calls **`/auth/check`** to see if the user is logged in.
- If logged in → shows **Dashboard + Logout**
- If logged out → shows **Login + Signup**
- After logout → updates the state and redirects to `/login`



And after login:-



Right now your `Navbar` only checks `localStorage` **once on page load**, so when you log in, it doesn't re-render to show **Logout** until you refresh.

We'll fix that by using a **shared login state** across your app — either with **React Context** (recommended) or a simple prop-based approach.

Let's go with **React Context**, which is clean, modern, and doesn't require heavy libraries like Redux.

Final Fix – Using React Context (Live Navbar Update)

✅ Step 1: Create a `AuthContext.js` file

📁 `src/context/AuthContext.js`

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

// Create context
export const AuthContext = createContext();

// Create provider
export const AuthProvider = ({ children }) => {
  const [loggedIn, setLoggedIn] = useState(localStorage.getItem('loggedIn')
=== 'true');

  useEffect(() => {
    // Sync with localStorage whenever it changes
    const stored = localStorage.getItem('loggedIn') === 'true';
    setLoggedIn(stored);
  }, []);

  const login = () => {
    localStorage.setItem('loggedIn', 'true');
    setLoggedIn(true);
  };

  const logout = () => {
    localStorage.removeItem('loggedIn');
    setLoggedIn(false);
  };

  return (
    <AuthContext.Provider value={{ loggedIn, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};
```

✅ Step 2: Wrap your App with the AuthProvider

📄 src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { AuthProvider } from './context/AuthContext';



const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <AuthProvider>
    <App />
  </AuthProvider>
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

✅ Step 3: Update `Navbar.js`

📁 src/components/Navbar.js

```
import React, { useContext } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import axios from 'axios';
import { AuthContext } from '../context/AuthContext';

const Navbar = () => {
  const { loggedIn, logout } = useContext(AuthContext);
  const navigate = useNavigate();

  const handleLogout = async () => {
    try {
      await axios.post('http://localhost:5000/auth/logout', {}, {
withCredentials: true });
      logout(); // update global state
      alert('Logged out successfully');
      navigate('/login');
    } catch (err) {
      console.error(err);
    }
  };

  return (
    <nav style={{ padding: '10px', borderBottom: '1px solid #ccc' }}>
      {!loggedIn ? (
        <>
          <Link to="/signup" style={{ marginRight: 10 }}>Signup</Link>
          <Link to="/login" style={{ marginRight: 10 }}>Login</Link>
        </>
      ) : (
        <>
          <Link to="/dashboard" style={{ marginRight: 10 }}>Dashboard</Link>
          <button onClick={handleLogout}>Logout</button>
        </>
      )}
    </nav>
  );
};

export default Navbar;
```

✅ Step 4: Update `Login.js`

📁 `src/components/Login.js`

```
import React, { useState, useContext } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import { AuthContext } from '../context/AuthContext';

const Login = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const navigate = useNavigate();
  const { login } = useContext(AuthContext);

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      await axios.post('http://localhost:5000/auth/login', { username,
password }, { withCredentials: true });
      login(); // update context immediately
      alert('Logged in successfully');
      navigate('/dashboard');
    } catch (err) {
      alert(err.response?.data || 'Login failed');
    }
  };

  return (
    <div>
      <h2>Login</h2>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          placeholder="Username"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
        /><br /><br />
        <input
          type="password"
          placeholder="Password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
        /><br /><br />
        <button type="submit">Login</button>
      </form>
    </div>
  );
};

export default Login;
```

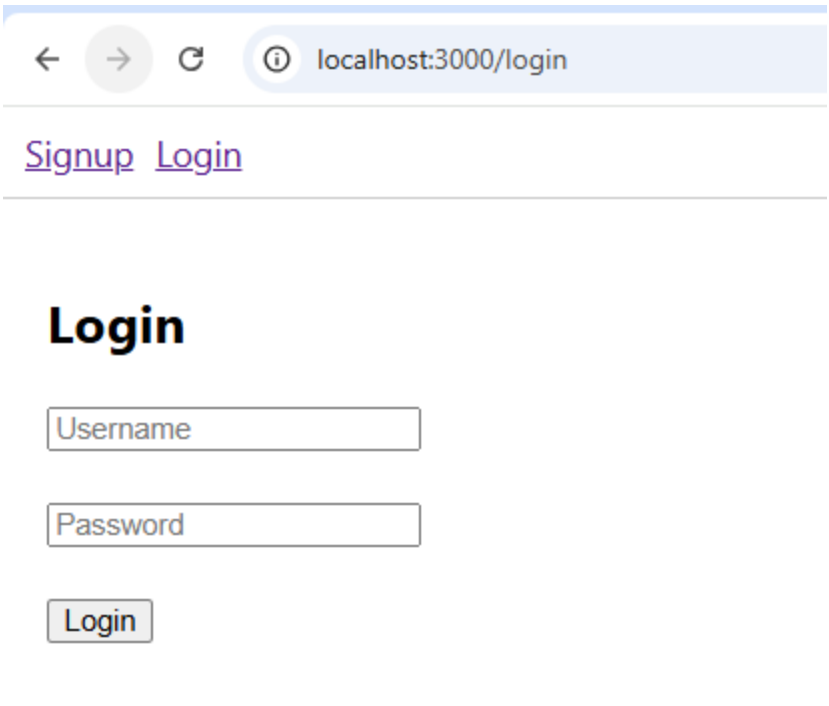
✅ Result

Now your app will:

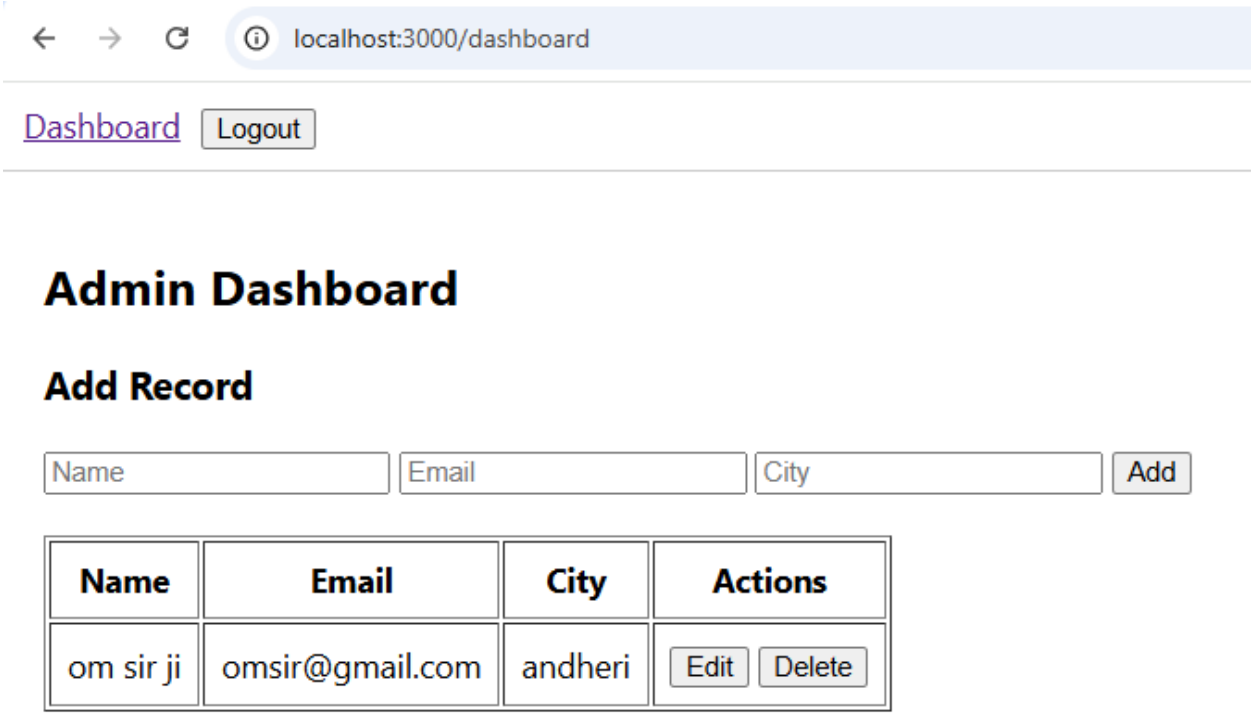
- Instantly update the navbar when login/logout happens — **no page refresh needed**.
- Automatically persist the login state using `localStorage`.
- Work cleanly with your existing backend (`express-session`).

Summary of Benefits:-

Feature	Without Context	With Context
Navbar updates instantly after login/logout	❌ Needs refresh	✅ Instant
Centralized login/logout logic	❌ Scattered	✅ Single source
Scalable for larger apps	❌ Harder	✅ Easy


localhost:3000/login
Signup Login
Login
Username
Password
Login

After login :-


localhost:3000/dashboard
Dashboard Logout
Admin Dashboard
Add Record
Name
Email
City
Add
Name
Email
City
Actions
om sir ji
omsir@gmail.com
andheri
Edit
Delete