

## Experiment No. 6

**Aim:** Implement **authentication and user roles** with JWT

**Code:**

### WeatherDashboard.jsx Code

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

const WeatherDashboard = () => {
  const [city, setCity] = useState("");
  const [weatherCards, setWeatherCards] = useState([]);
  const [currentLocationWeather, setCurrentLocationWeather] = useState(null);
  const [currentLocationForecast, setCurrentLocationForecast] = useState([]);
  const [error, setError] = useState("");
  const [user, setUser] = useState(null);
  const [history, setHistory] = useState(() => JSON.parse(localStorage.getItem("history")) || []);
  const navigate = useNavigate();
  const API_KEY = '*****UR_KEY*****';
  const getBackgroundImage = (weatherCondition) => {
<CODE>
  };

  const fetchCurrentLocationWeather = async (lat, lon) => {
    try {
      const res = await
    axios.get(`https://api.openweathermap.org/data/2.5/weather?lat=${lat}&lon=${lon}&appid=${API_K
    EY}&units=metric`);
      setCurrentLocationWeather(res.data);
      const forecastRes = await
    axios.get(`https://api.openweathermap.org/data/3.0/onecall?lat=${lat}&lon=${lon}&exclude=minute
    ly,hourly,alerts&appid=${API_KEY}&units=metric`);
      setCurrentLocationForecast(forecastRes.data.daily.slice(0, 7));
    } catch (err) {
      setError('Could not fetch your location's weather.');
```

```

if (navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(
    (pos) => fetchCurrentLocationWeather(pos.coords.latitude, pos.coords.longitude),
    () => setError('Permission denied for location access or location not found.')
  );
} else setError('Geolocation is not supported by this browser.');
```

const token = localStorage.getItem("token");

```

if (!token) {
  // navigate("/login");
}
fetchDashboard();
}, []);

const fetchCityWeather = async (e) => {
  e.preventDefault();
  if (!city.trim()) return;
  try { const res = await
axios.get(`https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${API_KEY}&units=metric`); const forecastRes = await
axios.get(`https://api.openweathermap.org/data/2.5/forecast?q=${city}&appid=${API_KEY}&units=metric`) const dailyForecast = forecastRes.data.list.filter((reading) =>
  reading.dt_txt.includes("12:00:00")
).slice(0, 5);
const newCard = { ...res.data, forecast: dailyForecast };
const alreadyExists = weatherCards.some((w) => w.name === newCard.name) ||
currentLocationWeather?.name === newCard.name;
if (alreadyExists) return setError('City is already shown.');
```

setWeatherCards((prev) => [...prev, newCard]);

```

setHistory((prev) => {
  const updated = [...new Set([newCard.name, ...prev])].slice(0, 5);
  localStorage.setItem("history", JSON.stringify(updated));
  return updated;
});
setCity("");
setError("");
} catch (err) {
  setError('City not found.');
```

}

```

};

const getWeatherIcon = (iconCode) =>
`https://openweathermap.org/img/wn/${iconCode}@2x.png`;

return (
  </div>
  { /* Right Section - Search, Details & Forecast (Scrollable) */ }
  <div className="md:col-span-1 lg:col-span-2 bg-gradient-to-br from-gray-100 to-gray-200 text-gray-800 p-6 md:p-8 flex flex-col z-20 overflow-y-auto">
    <h2 className="text-4xl font-semibold text-center mb-6 text-indigo-700">Other Cities</h2>
    { /* Search Bar */ }
  </div>
);

```

```

    <form onSubmit={fetchCityWeather} className="flex items-center space-x-3 mb-6 bg-white
rounded-full p-2 shadow-lg">
      <input
        type="text"
        value={city}
        onChange={(e) => setCity(e.target.value)}
        placeholder="Search any city"
        className="flex-grow bg-transparent text-lg px-2 text-gray-800 placeholder-gray-500
focus:outline-none"/>
      <button type="submit" className="bg-indigo-500 text-white p-3 rounded-full hover:bg-indigo-
600 transition duration-300 transform hover:scale-105">
        <svg xmlns="http://www.w3.org/2000/svg" className="h-6 w-6" fill="none" viewBox="0 0 24
24" stroke="currentColor">
          <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M21 21l-6-6m2-
5a7 7 0 11-14 0 7 7 0 0 1 14 0z" />
        </svg>
      </button>
    </form>

    {error && <p className="text-red-500 text-center mb-4">{error}</p>}
      </div>
    </div>
  )}
</div>
</div>
);
};

```

```
export default WeatherDashboard;
```

### AdminDashboard.jsx

```

import axios from 'axios';
import React, { useEffect, useState } from 'react';
import { useNavigate } from 'react-router-dom';
const AdminDashboard = () => {
  const [users, setUsers] = useState([]);
  const navigate = useNavigate()
  const [isAdmin, setAdmin] = useState(false);

  const fetchDashboard = async () => {
    const token = localStorage.getItem("token");
    try {
      const res = await axios.get("http://localhost:5000/adminDashboard", {
        headers: { Authorization: `Bearer ${token}` }
      });
      setUsers(res.data.users);

      if (res.data.admin.role !== "admin") {
        navigate("/weatherDashboard");
      }
    }
  }

```

```

    navigate("/adminDashboard")
  } catch (err) {
    console.log("Access denied", err);
    navigate("/weatherDashboard"); // optional: redirect if not admin
  }
};
useEffect(() => {
  fetchDashboard();
}, []);
return (
  <div className="min-h-screen bg-gradient-to-r from-blue-100 via-purple-100 to-pink-100 px-4 p-6">
    <div className="max-w-5xl mx-auto">
      </* Dashboard Header */>
      <div className="mb-8">
        <h1 className="text-3xl font-bold text-indigo-700 mb-2">Admin Dashboard</h1>
        <p className="text-gray-600">Overview of logged-in users</p>
      </div>

      </* Stats Box */>
      <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4 mb-6">
        <div className="bg-white shadow rounded-2xl p-6 text-center">
          <h2 className="text-xl font-semibold text-gray-800">Total Logged-in Users</h2>
          <p className="text-4xl text-indigo-600 mt-2">{users.length}</p>
        </div>
      </div>

      </* User Table */>
      <div className="bg-white shadow-md rounded-2xl overflow-x-auto">
        <table className="min-w-full table-auto text-sm text-gray-700">
          <thead className="bg-indigo-50">
            <tr>
              <th className="px-6 py-3 text-left font-semibold">#</th>
              <th className="px-6 py-3 text-left font-semibold">Name</th>
              <th className="px-6 py-3 text-left font-semibold">Email</th>
              <th className="px-6 py-3 text-left font-semibold">Role</th>
              <th className="px-6 py-3 text-left font-semibold">Login Time</th>
            </tr>
          </thead>
          <tbody>
            {users.map((user, idx) => (
              <tr key={user.id} className="border-b hover:bg-gray-50">
                <td className="px-6 py-3">{idx + 1}</td>
                <td className="px-6 py-3">{user.name}</td>
                <td className="px-6 py-3">{user.email}</td>
                <td className="px-6 py-3">{user.role}</td>
                <td className="px-6 py-3">{user.lastLogin}</td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    </div>
  </div>

```

```

    </div>
  </div>
);
};
export default AdminDashboard;

```

### authMiddleware.jsx

```

const jwt = require("jsonwebtoken");
const SECRET = "ABCD@1234";

function authMiddleware(req,res,next){
  const authHeader = req.headers.authorization;
  if (!authHeader) return res.status(401).json({ message: "No token provided" });

  const token = authHeader.split(" ")[1]; // "Bearer <token>"


  jwt.verify(token, SECRET, (err, user) => {
    if (err) return res.status(403).json({ message: "Invalid token" });
    req.user = user;
    next();
  });
}
module.exports = authMiddleware;


```

### Index.jsx

```

require("dotenv").config(); //  Load .env first
const express = require('express');
const cors = require("cors");
const app = express();
const mongoose = require("mongoose");
const jwt = require("jsonwebtoken");
const bcrypt = require("bcryptjs");
const authenticateToken = require('./middleware/authMiddleware');
const authorizeRole = require('./middleware/authorizeRole');
//  Use environment variables
const PORT = process.env.PORT || 5000;
const SECRET = process.env.JWT_SECRET || "ABCD@1234";
const MONGO_URI = process.env.MONGO_URI;

//  Connect to MongoDB Atlas
mongoose.connect(MONGO_URI)
  .then(() => console.log("  Connected to MongoDB Atlas"))
  .catch(err => console.error("  MongoDB connection error:", err));

//  User Schema & Model
const userSchema = new mongoose.Schema({
  name: String,
  email: { type: String, unique: true },
  hashedPass: String,

```

```

    role: { type: String, default: "user" },
    lastLogin: String
  });
const User = mongoose.model("User", userSchema);
// ✅ Middleware
app.use(express.urlencoded({ extended: true }));
app.use(express.json());

var corsOptions = {
  origin: 'http://localhost:3000',
  optionsSuccessStatus: 200,
  credentials: true,
};
app.use(cors(corsOptions));

// ✅ SIGNUP
app.post("/signup", async (req, res) => {
  <CODE>
});

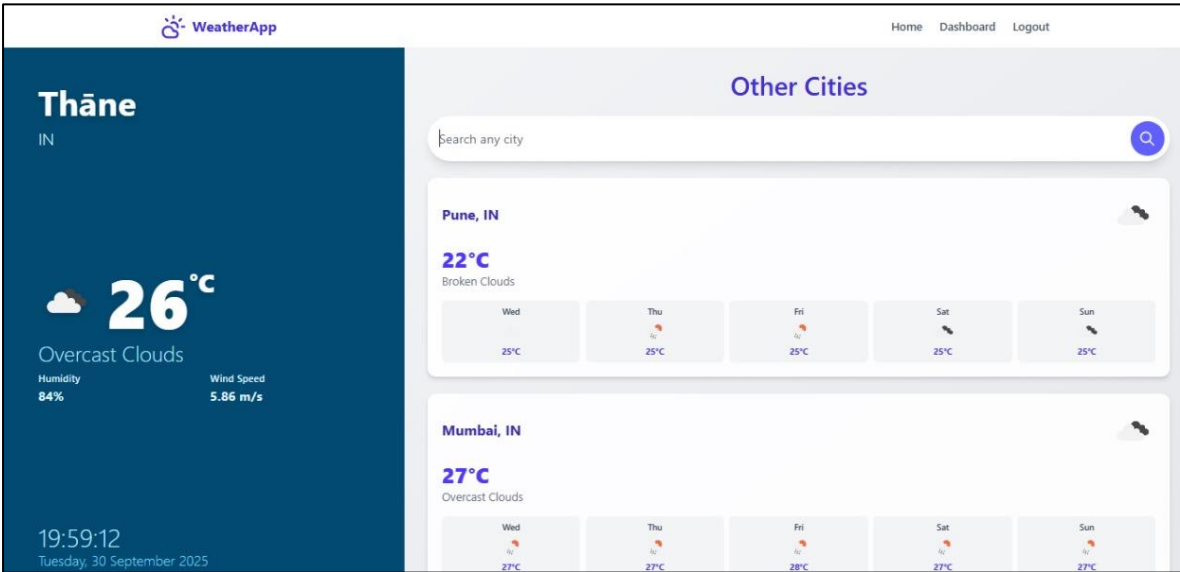
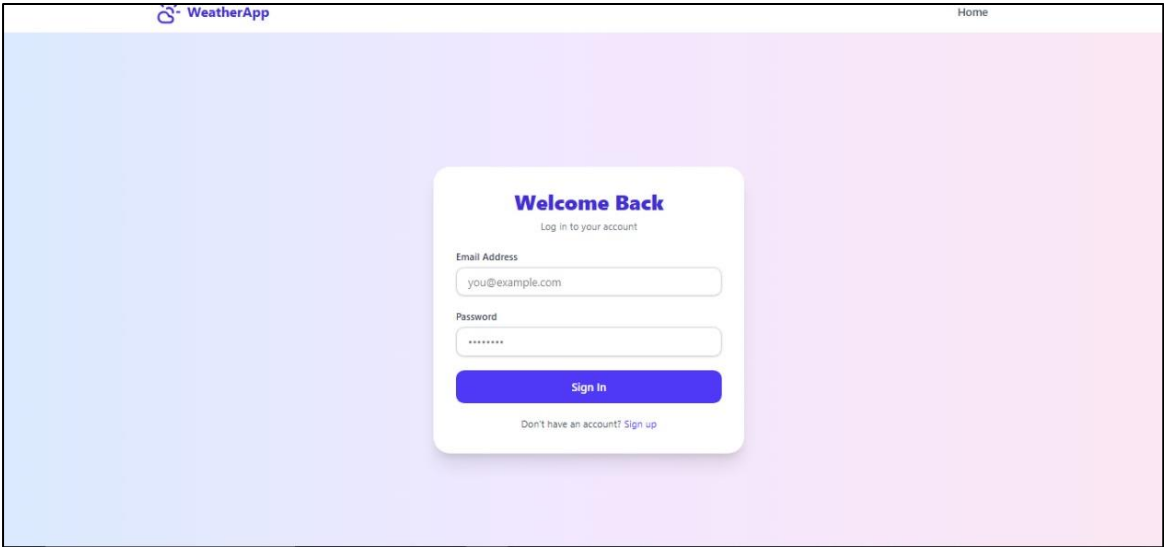
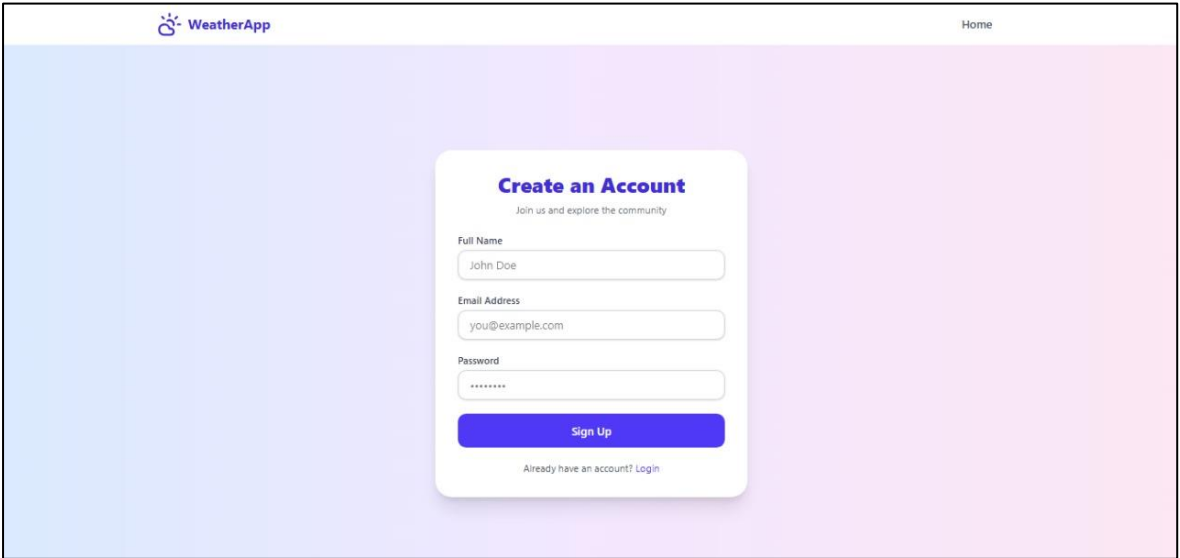
// ✅ LOGIN
app.post("/login", async (req, res) => {
  try {
    const { email, password } = req.body;

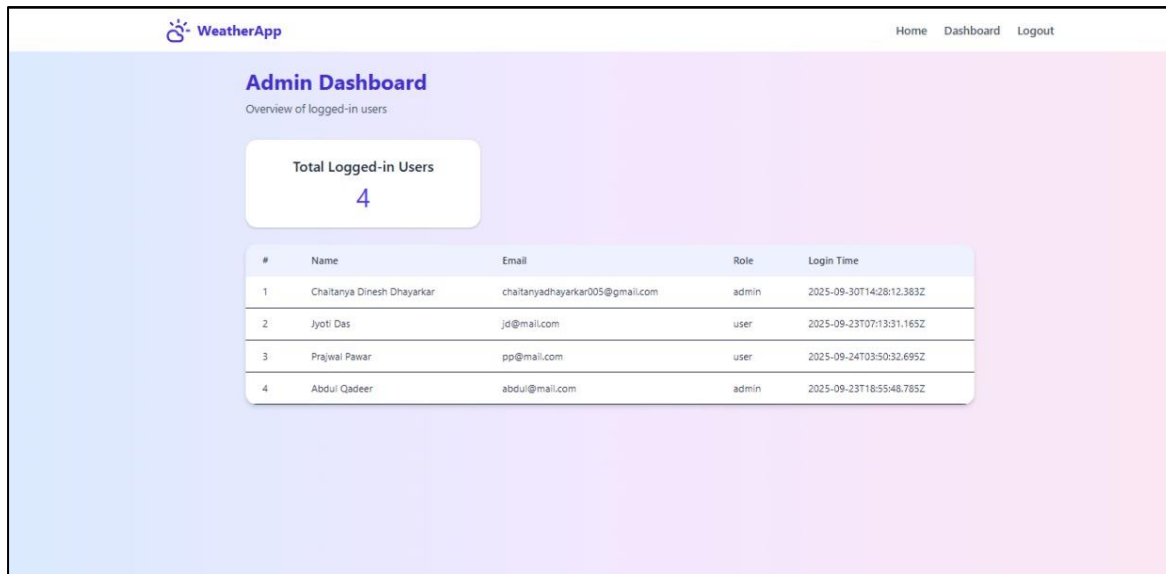
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(401).json({ message: "Invalid credentials", status: false });
    }
    // Update last login time
    user.lastLogin = new Date().toISOString();
    await user.save();
    // Generate JWT
    const token = jwt.sign({ id: user._id, role: user.role }, SECRET, { expiresIn: "1d" });
    res.status(200).json({
      message: "Login successful",
      token,
      role: user.role,
      lastLogin: user.lastLogin,
      status: true
    });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// ✅ DASHBOARD
app.get("/dashboard", authenticateToken, async (req, res) => {
  try {
    const user = await User.findById(req.user.id);
    if (!user) {
      return res.status(404).json({ message: "User not exists" });
    }
    res.json({ message: "Welcome to dashboard", user });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

```

Output:





### Conclusion:

I have successfully executed and implemented authentication and user roles with **JWT**.