 **Add Realistic Earth Textures & Space Background**

* Replace the plain sphere with a high-resolution Earth texture.
* Load a starry sky texture as the scene background to simulate space.
* This will immediately make your globe feel more polished and visually appealing.

 **Implement City Markers & Click Events**

* Convert latitude/longitude to 3D coordinates on the globe.
* Place small markers (red dots, pins, or custom icons) for different cities.
* On marker click, fetch weather, traffic, and news for that city and display it in the overlay.

 **Fetch All Relevant Data**

* Right now, you’re only fetching weather data.
* Extend your backend to fetch traffic and news for the same city.
* Consolidate all data (weather, traffic, news) into one overlay for a seamless user experience.

 **Performance Optimization**

* Use memoization (e.g., React.memo) and caching if you have many markers or large textures.
* Implement lazy loading if certain data is only needed when a user interacts with a marker.

 **UI/UX Enhancements**

* Add transitions, hover effects, or animations for markers.
* Make the overlay design more “Netflix-level” with sleek styling and smooth animations.
* Ensure mobile responsiveness so it looks good on all devices.

 **Deployment to AWS**

* Showcase your cloud skills by deploying both your backend (Node.js) and frontend (React) on AWS.
* For example, use AWS Amplify for the React app and AWS EC2 or Elastic Beanstalk for the backend.

 **Testing & QA**

* Add basic tests (e.g., Jest or React Testing Library) to ensure each component works as intended.
* Manually test various lat/lon inputs to confirm data fetches correctly.

 **Documentation & Polish**

* Write clear instructions in your README for setting up and running the project.
* Create a short demo video or GIF to show the interactive globe in action.

 **Showcase on LinkedIn/GitHub**

* Once deployed, share your live URL.
* Highlight the tech stack and your unique contributions.