**Latency Visualizer App – Technical Overview**

This document provides a structured overview of the Latency Topology Visualizer application, explaining its architecture, main components, supporting utilities, and design rationale.

**Application Structure**

The codebase is organized in a modular and scalable manner, where each folder serves a specific function:

* App Folder: Contains global configuration including routing and base styles.
* Components Folder: Includes reusable UI elements like the 3D globe, charts, and control panels.
* Hooks Folder: Houses custom React hooks that handle data fetching and state management logic.
* Lib Folder: Provides utility functions for network checks, latency calculations, and other shared helpers.
* Data Folder: Stores mock data used for testing or fallback rendering.
* Types Folder: Defines TypeScript interfaces and types to ensure consistent data handling throughout the application.

We leverage centralized tools for latency tracking and share state updates via context or global stores. This promotes maintainability and performance efficiency.

## Main Features and Components

1. 3D Earth Globe (MapboxGlobe.tsx)

* Renders an interactive 3D globe using Mapbox and Three.js.
* Displays dynamic connections between regions based on network latency.
* Allows marker interaction for region-specific insights.
* Supports light and dark themes.

1. Swipe-Up Mobile Menu (MobileControlPanel.tsx)

* Mobile-optimized drawer with control options.
* Built with gesture-based interactivity.
* Provides a native-app-like experience.

1. Historical Latency Chart (HistoricalChart.tsx)

* Visualizes latency over different time ranges (hour/day/week).
* Supports cryptocurrency-based filtering.
* Displays statistical insights such as minimum, maximum, and average latency.

1. Network Topology Map (NetworkTopology.tsx)

* Displays network paths and routing hops.
* Includes filtering and search functionality.
* Provides detailed connection metrics per route.

## Custom Hooks

* useRealTimeLatency.ts: Continuously fetches and updates latency data, and computes statistical values.
* useStore.ts: Manages UI state and user selections using Zustand.
* useTheme.ts: Synchronizes the application’s appearance with system-level theme preferences.

## API Interfaces

* LatencyMonitor: Measures round-trip time (RTT) to various nodes. Fallbacks to mock data when live measurements fail.
* NetworkPerformanceMonitor: Assesses client-side network health, including bandwidth type and RTT.

## Performance Optimizations

* Selective component re-rendering to reduce unnecessary DOM updates.
* Efficient re-use of 3D map markers to maintain smooth visuals.
* Animation lifecycle control to prevent memory leaks and CPU strain.

## Error Handling and Resilience

* Gracefully degrades using mock data when APIs fail.
* Component-level error boundaries ensure isolated failures don’t affect the full UI.
* Retry mechanisms for transient network issues.