Plus Points in Implementation (Overall Evaluation Criteria)

1. **UI Component Architecture:**
    - Design a modular, reusable component (e.g., atomic design principles).
    - Separate presentational components from logic/container components.

2. **Responsiveness & Accessibility:**
    - Ensure the application is fully responsive across mobile, tablet, and desktop breakpoints.
    - Implement accessibility standards (WCAG 2.1) — ARIA labels, keyboard navigation, color contrast.

3. **Performance Optimization:**
    - Apply lazy loading, code splitting, and memoization to minimize render cycles.
    - Profile and reduce Largest Contentful Paint (LCP) and Cumulative Layout Shift (CLS).

4. **State Management:**
    - Choose an appropriate state management strategy (Context API, Redux, Zustand, etc.).
    - Clearly distinguish local component state from global/shared state.

5. **Error & Loading States:**
    - Handle network failures, empty states, and skeleton loaders gracefully.
    - Provide clear, user-friendly error messages and retry mechanisms.

6. **Unit Testing:**
    - Write unit tests for components (React Testing Library / Jest).
    - Add integration or E2E tests for critical user flows (Cypress / Playwright).

7. **API Integration & Data Fetching:**
    - Use REST or WebSocket APIs effectively; show real-time updates where needed.
    - Implement proper caching (React Query, SWR, or manual) to reduce redundant API calls.

8. **Design Consistency:**
    - Follow a consistent design language / design system (MUI, Tailwind, custom tokens).
    - Use consistent typography, spacing, and color palette throughout the application.

Instructions:

1. **Read and Understand the Problem Statement:**
    - Carefully read the problem statement. Understand the UI requirements, user journeys, expected interactions, and any constraints mentioned.

2. **Choose Your Frontend Stack:**
    - Select a framework you are comfortable with (React, Vue, Angular, or plain JS). React is preferred for most tasks at MoveInSync.
    - You may use any supporting libraries (state management, charting, maps, etc.) as needed.

3. **Design Your UI/UX:**
    - Sketch or wireframe your UI before writing code. Think about layout, user flows, and component hierarchy.
    - Focus on usability — your interface should be intuitive and accessible.

4. **Write the Code:**
    - Implement your solution following best practices: clean component structure, meaningful variable/prop names, and comments where necessary.
    - Break down the UI into smaller, reusable components to improve readability and maintainability.

5. **Test Your Application:**
    - Test across different screen sizes and browsers. Include edge cases: empty states, error responses, slow networks.
    - Ensure your application produces correct visual output and interactions for all scenarios.

6. **Document Your Code:**
    - Add a README with setup instructions, a brief explanation of your architecture, and any design decisions made.
    - Comment complex logic or non-obvious UI behavior.

7. **Submit Your Solution:**
    - Submit your code on GitHub and share the repository link. Include a live demo link if possible (Vercel, Netlify, etc.).

8. **Demonstration:**
    - Include a short screen recording or walkthrough video showcasing the key features of your implementation.

Real-Time Vehicle Tracking & Geofence Visualization UI

Context

> *MoveInSync operates enterprise transportation for hundreds of corporate clients. Vehicles stream GPS coordinates in real time. Operations teams need a live map interface to track all active trips, visualize geofences, monitor alerts, and take action — all from a single command center screen.*

Problem Statement

Design and implement a frontend Live Vehicle Tracking application that provides:

- A real-time interactive map with all active vehicles.
- Geofence visualization and event highlighting.
- A side-panel command center for trip monitoring.

I. Live Map Interface

A. Interactive Map View

The core of the application is a real-time map (using Leaflet, Google Maps, or Mapbox):

- Vehicle markers that update position in real time via WebSocket
- Marker icons reflecting trip status: idle (grey), in-progress (green), delayed (amber), alert (red)
- Click a vehicle marker to open a popup with: Driver name, Trip ID, Current speed, ETA to next stop
- Smooth marker animation between GPS updates (no teleporting)

B. Geofence Overlay

Geofences must be drawn on the map:

- Office geofences rendered as filled circles or polygons with a distinct color (e.g., blue)
- Pickup geofences rendered as smaller circles (e.g., orange)
- Geofences should pulse or highlight when a vehicle enters them
- Toggle to show/hide geofence overlays without losing map state

C. Trip Route Visualization

When a specific trip is selected:

- Draw the planned route as a polyline on the map
- Show completed route segment in a darker color vs. remaining route
- Display pickup stop markers with employee names
- Show current vehicle position relative to the route

II. Command Center Side Panel

A. Trip Summary Panel

A collapsible side panel showing:

- Total active trips, delayed trips, trips inside office geofence, trips approaching pickup
- Filter trips by: region, office, route, status, time window
- List of filtered trips with basic info: Trip ID, Driver, Status, ETA
- Click a trip in the list to pan the map to that vehicle and open its detail

B. Real-Time Alert Feed

A live alert ticker at the top or side:

- Geofence entry/exit events, speeding alerts, manual trip closure discrepancies
- Each alert clickable — pans map to the relevant vehicle
- Auto-dismiss old alerts; keep unread count badge

III. Trip Lifecycle & Geofence Events

A. Geofence Event Notification

When a vehicle crosses a geofence boundary:

- Flash the affected vehicle marker and geofence overlay briefly
- Show a toast notification: 'Vehicle [ID] entered Office Geofence – Trip auto-closed'
- Update trip status in the side panel immediately

B. Employee Pickup Notification View

When a vehicle arrives at a pickup geofence:

- Show a subtle card notification on the map: 'Cab arrived for [Employee Name]'
- Marker briefly bounces or pulses
- Notification auto-clears after 30 seconds

IV. Historical Playback (Bonus Feature)

An optional bonus feature for trip replay:

- Select a completed trip and play back its GPS path on the map
- Playback speed control: 1x / 5x / 10x
- Scrub bar to jump to any point in the trip
- Show geofence events along the timeline

Expected Discussion Areas

1. WebSocket & Real-Time Rendering

How do you manage hundreds of simultaneously updating vehicle markers without performance degradation? Discuss debouncing position updates, rendering strategies (canvas vs. DOM), and WebSocket reconnection logic.

2. Map Library Choice

Compare Leaflet, Mapbox GL JS, and Google Maps for this use case. What factors (bundle size, tile costs, offline support, custom styling) drive your recommendation?

3. Geofence Rendering Performance

If there are 500 office geofences and 2,000 pickup geofences, how do you render them efficiently? Consider tile-based approaches and level-of-detail (LOD) techniques.

4. Offline / Degraded Network Handling

If the WebSocket drops, how does the UI communicate stale data to the operator? What is the re-connection strategy, and how are buffered location updates replayed?

Real-Time Vehicle Tracking & Geofence Visualization UI