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.

Driver Feedback & Sentiment Dashboard

Context

MoveInSync collects post-trip feedback from employees. Currently, feedback is stored but not surfaced meaningfully to ops teams or employees. The company wants a configurable, intuitive feedback experience for riders, and a rich analytics dashboard for administrators to track driver sentiment trends in real time.

Problem Statement

Design and implement a frontend application that provides:

- A configurable post-trip feedback form for employees.
- A real-time admin analytics dashboard visualizing driver sentiment data.
- Feature-flag-driven visibility of feedback modules.

I. Configurable Feedback Form UI

A. Multi-Entity Feedback Support

The feedback form must support feedback for multiple entities. Each entity type should be togglable via feature flags (no code change required to enable/disable):

- Driver (rating, text, specific attribute tags)
- Trip (punctuality, route accuracy, comfort)
- Mobile App (UX, speed, reliability)
- Marshal (safety, helpfulness, professionalism)

UI requirements per entity:

- Star rating component (1–5 with hover states)
- Tag-based quick feedback chips (e.g., 'Rash Driving', 'Very Polite')
- Optional free-text area with character count
- Clear section headers identifying which entity is being rated

B. Feature Flag Configuration

The form should read a configuration object (from API or local config) to dynamically render only enabled feedback sections:

Example: { driverFeedback: true, tripFeedback: true, appFeedback: false, marshalFeedback: false }

UI behavior:

- Disabled sections must NOT appear in the form at all
- Config changes should update the form without requiring a page reload
- Show a 'No feedback options available' empty state if all flags are off

C. UX & Interaction Details

The form must meet the following interaction standards:

- Inline validation — show errors immediately on blur, not just on submit
- Progress indicator if form has multiple steps
- Submission confirmation screen / success toast
- Prevent duplicate submissions (disable button after first click, show loading state)
- Mobile-first design: touch-friendly tap targets, responsive layout

II. Admin Sentiment Analytics Dashboard

A. Overview Panel

The dashboard landing view should show:

- Total feedback received (today / 7 days / 30 days — date range toggle)
- Overall sentiment distribution: Positive / Neutral / Negative (donut or pie chart)
- Average sentiment score across all drivers
- Count of drivers currently below the alert threshold

B. Driver Leaderboard

A sortable, filterable table of drivers with:

- Driver name, ID, total trips, average score, trend indicator (↑↓ vs last week)
- Color-coded rows: green (≥ 4.0), amber (2.5–3.9), red (< 2.5)
- Click-to-expand row showing recent 5 feedback entries for that driver
- Column sorting and text search / filter by score range

C. Feedback Timeline View

A chronological feed of recent feedback submissions:

- Show entity type, sentiment badge, score, timestamp, and truncated text
- Infinite scroll or paginated view
- Filter by: entity type, sentiment, date range, driver

D. Driver Detail Page

On clicking any driver from the leaderboard, open a detail view with:

- Sentiment score trend line chart over the last 30 days
- Breakdown by feedback tag (bar chart: 'Rash Driving' × 12, 'Very Polite' × 34 ...)
- Full feedback history table with pagination
- Alert badge if driver is currently flagged

E. Alert Notification UI

When a driver's average drops below threshold:

- Show an in-app notification banner or toast with driver name and current score
- Alerts list accessible from a bell icon in the nav with unread count badge
- Each alert should link directly to the driver's detail page

Expected Discussion Areas

1. Component Architecture

How are the feedback form and dashboard broken into reusable components? Walk through the component tree and explain prop flow vs. shared state.

2. State Management

Which state lives locally in components vs. in a global store? How is feature flag configuration propagated? How is real-time sentiment data kept fresh?

3. Data Visualization Choices

Justify the chart types chosen (line, donut, bar). How does the dashboard handle large datasets without performance degradation? Consider virtualized lists for the feedback timeline.

4. Accessibility & Responsiveness

How does the feedback form behave on mobile? Are star ratings keyboard-accessible? Does the dashboard reflow gracefully on smaller screens or become a read-only summary?

5. Backend Integration

Define the API contracts your frontend expects. How are loading and error states handled? What is the polling or WebSocket strategy for real-time score updates?

Driver Feedback & Sentiment Dashboard