## Scenario / Context

You are a frontend engineer on an EdTech platform tasked with developing a **Learning Continuity Matrix**, a feature designed to foster persistent student engagement through gamified progress tracking. This module integrates into the existing student dashboard, encouraging daily learning habits by visualizing progress and rewarding consistency. Your task is to implement a robust, type-safe frontend solution that balances interactivity, performance, and extensibility.

## Functional Requirements

### 1. Weekly Engagement Matrix

* **Objective**: Render a horizontal, week-based visualization of learning activity (Monday to Sunday).
* **Details**:
  + Each day displays:
    - The date in a localized format (e.g., "Mon, Oct 10").
    - A visual indicator of whether the daily learning objective was met (use mock data).
  + Interaction: Clicking a day toggles its completion status in an in-memory state, with optimistic updates and a brief animation (e.g., fade or scale).
  + Accessibility: Ensure keyboard navigation and screen reader compatibility for toggling.

### 2. Dynamic Motivation Panel

* **Objective**: Display a card summarizing the student's engagement metrics and tailored feedback.
* **Details**:
  + Metrics:
    - **Current Continuity**: Number of consecutive days with completed objectives, calculated dynamically.
    - **Peak Continuity**: Longest streak of completed days, derived from mock data history.
  + Feedback:
    - Provide a context-sensitive motivational message based on the current streak (e.g., "Impressive! You've maintained a 4-day streak!" or "Let's reignite your progress to surpass your 6-day record!").
    - Messages should vary based on streak length thresholds (e.g., 0-2 days, 3-5 days, 6+ days).
  + Persist the message state briefly across re-renders for a smooth user experience.

### 3. Peer Benchmarking (Optional but Recommended)

* **Objective**: Showcase a leaderboard highlighting top-performing students.
* **Details**:
  + Display up to 3 students with their current streak counts, sorted by streak length.
  + Highlight the logged-in student (mocked as studentId = 101) with a distinct visual style.
  + Ensure the leaderboard updates dynamically if streak data changes.
  + Use mock data for other students.

### 4. State Persistence Controls

* **Objective**: Provide controls to persist or reset the engagement data.
* **Details**:
  + **Persist**: Log the current state of the engagement matrix to the console in a structured JSON format.
  + **Reset**: Revert to the initial mock data state, with a confirmation prompt to prevent accidental resets.

## Technical Constraints

* **Tech Stack**: Use React with TypeScript exclusively.
* **Component Structure**:
  + Employ functional components only; avoid class-based components.
  + Decompose into modular components: EngagementMatrix, MotivationPanel, PeerBenchmark (if implemented).
  + Use component composition and prop-drilling or context for state sharing where appropriate.
* **State Management**:
  + Leverage useState for local state and useEffect for side effects (e.g., recalculating streaks or logging).
  + Implement at least one custom hook to encapsulate logic (e.g., streak calculation or data persistence).
* **Typing**:
  + Define precise TypeScript interfaces for:

interface EngagementEntry {

date: string; // ISO 8601 format (YYYY-MM-DD)

isCompleted: boolean;

}

interface PeerEntry {

id: number;

displayName: string;

continuityCount: number;

}

* + Ensure all props and state are strongly typed.
* **Mock Data**:
  + Use in-memory mock data; no external API calls.
  + Example data structure:

const mockEngagement: EngagementEntry[] = [

{ date: "2025-10-06", isCompleted: true },

{ date: "2025-10-07", isCompleted: false },

// ... other days

];

const mockPeers: PeerEntry[] = [

{ id: 101, displayName: "Student 1", continuityCount: 3 },

{ id: 102, displayName: "Student 2", continuityCount: 5 },

// ... other peers

];

* **Styling**:
  + Use a modular styling approach (e.g., CSS Modules, styled-components, or Tailwind CSS).
  + Ensure the design is cohesive with a modern dashboard aesthetic (clean typography, subtle animations, responsive layout).
  + Support responsive breakpoints for desktop and tablet views.
* **Performance**:
  + Optimize re-renders using React.memo or useMemo where applicable.
  + Ensure state updates are efficient to handle frequent toggling.

## Evaluation Criteria

1. **Code Integrity**:
   * Modular, reusable components with clear separation of concerns.
   * Strict TypeScript adherence with no any types or implicit type coercion.
   * Semantic naming for variables, components, and hooks.
2. **State Logic**:
   * Accurate toggling of engagement status with optimistic updates.
   * Correct computation of current and peak continuity metrics.
   * Robust handling of edge cases (e.g., empty data, non-consecutive streaks).
3. **User Experience**:
   * Intuitive and visually engaging matrix display with smooth interactions.
   * Dynamic, context-aware motivational feedback.
   * Accessible and responsive design compatible with dashboard integration.
4. **Scalability**:
   * Codebase structured for easy integration into a larger dashboard.
   * Extensible for future features (e.g., monthly matrix view, cross-student analytics).
   * Minimal technical debt with clear documentation or self-explanatory code.

## Additional Notes

* Prioritize a11y (accessibility) for all interactive elements.
* Avoid external libraries for core functionality; focus on React primitives.
* Test your implementation with edge cases (e.g., no completed days, streaks spanning multiple weeks).
* Ensure the solution is performant for up to 30 days of engagement data.