Embeddable Credit Score Simulator SDK

Duration: 3 days

Objective: Build a white-labeled React/TypeScript SDK that provides a credit score simulator, enabling users to explore interactive scenarios affecting their credit health.

Core Requirements

1. Credit Simulation Engine

- Implement an interactive simulation engine with the following controls:
    - **Credit Utilization Slider**: Allows users to adjust their credit utilization percentage (0% to 100%).
    - **Payment History Toggle**: Lets users toggle between "On-time Payments" and "Missed Payments."
    - **New Credit Applications Counter**: Tracks how many new credit applications the user has made.
    - **Credit Age Slider**: Lets users adjust the average age of their credit accounts (in years).
    - **Debt-to-Income Ratio Input**: Allows users to input their monthly debt payments relative to their income.
- **Simulation Logic**:
    - The SDK should calculate a mock credit score based on the user's inputs.
    - Ensure the score reflects realistic impacts:
        - High credit utilization negatively affects the score.
        - Missed payments have a significant negative impact.
        - Multiple new credit applications slightly reduce the score.

        - Older credit accounts positively impact the score.
        - A high debt-to-income ratio negatively impacts the score.
- **Credit Score Ranges**:
    - **300-579 (Poor)**: Likely to have difficulty getting approved for credit.
    - **580-669 (Fair)**: May qualify for credit but with higher interest rates.
    - **670-739 (Good)**: Likely to qualify for credit with favorable terms.
    - **740-799 (Very Good)**: Likely to qualify for the best interest rates.
    - **800-850 (Excellent)**: Exceptional creditworthiness.
- **Edge Case**:
    - Handle scenarios where users input extreme values (e.g., 200% credit utilization or negative values).
    - Ensure the simulation remains stable and provides meaningful feedback.

---

2. White-Label Implementation

- Create a theme system that allows banks to customize the SDK's appearance and behavior.
    - **Score Ranges**: Banks should be able to define custom score ranges (e.g., Poor, Fair, Good, Excellent) and assign brand-specific colors to each range.
    - **Component Customization**: Allow banks to replace default icons, fonts, and other UI elements with their own branded assets.
    - **Localization**: Support locale-specific formatting for numbers, dates, and currencies.
- **Edge Case**:
    - Ensure the SDK gracefully handles incomplete or invalid theme configurations (e.g., missing colors or fonts).
    - Provide fallback styles to maintain usability even if the host site's CSS conflicts with the SDK.

---

3. Cross-Framework Embedding

- The SDK should be embeddable in multiple environments:
  - **Modern React**: Support React 18+ with Next.js App Router.
  - **Legacy Stacks**: Ensure compatibility with vanilla JavaScript and jQuery-based websites.
- **Edge Case**:
  - Handle scenarios where the host site uses conflicting versions of React or other libraries.
  - Ensure the SDK does not break if the host site has global CSS styles that conflict with the SDK's styles.

Deliverables

1. **SDK Code**:
   - Core simulation logic and UI components.
   - Theme configuration system.
2. **Deployed SDK**:
   - A modern React implementation.
   - A live deployment of the SDK for easy access and testing
3. **5-Minute Video Walkthrough**:
   - Explain key architectural decisions and tradeoffs.
   - Highlight any areas for improvement that fall outside the current scope.

Evaluation Criteria

✅ **SDK Design**:

- Clean separation of concerns between business logic and UI.
- Effective handling of edge cases (e.g., invalid inputs, conflicting styles).

✅ **White-Labeling**:

- Flexibility and extensibility of the theme system.
- Consistent rendering across different host environments.

✅ **Financial Relevance**:

- Pedagogical value of the simulation engine.
- Clear and intuitive data visualization.
