**Part 2: FRONTEND CODE GENERATION PROMPT**

**Project Overview (Frontend Focus):**

Develop the frontend for the Advanced Pilot Training Platform—a next-generation flight training management system that integrates advanced scheduling, syllabus building, document management, adaptive assessments, real-time analytics, immersive 3D/AR visualizations, and enhanced collaboration tools. The frontend will be built using Next.js and React with TypeScript, following best practices for responsive design, accessibility, and performance. This layer communicates with the backend via secure RESTful APIs and supports offline capabilities through Progressive Web App (PWA) features.

The repository structure for the frontend is as follows:

/advanced-pilot-training-platform

/frontend

/components # Reusable UI components (buttons, forms, modals, charts)

/pages # Page-level components integrating core functionalities

/hooks # Custom React hooks for data fetching, real-time updates, adaptive UI

/services # API integrations with backend modules (document, syllabus, assessment, scheduling, etc.)

/styles # Global styles, Tailwind CSS configuration, MUI themes

/assets # Static assets (images, icons, 3D models)

/visualizations # Components for 3D/AR views, interactive dashboards, data overlays

/collaboration # UI components for real-time collaboration and smart workspaces

**Instructions:**

1. **Component Library Foundation:**
   * Create a comprehensive set of base UI components:
     + Standardized elements (Buttons, Inputs, Selects) with accessibility (ARIA) and responsive design.
     + Form components integrated with validation libraries (React Hook Form, Yup).
     + Data visualization components using Recharts, D3.js, and Three.js for interactive 3D/AR views.
     + Navigation elements (tabs, breadcrumbs, pagination) and feedback components (alerts, notifications, modals).
2. **Syllabus Builder Interface:**
   * Develop a robust, drag-and-drop Syllabus Builder component:
     + Tree-view interface displaying modules, lessons, and exercises.
     + Inline editing for syllabus elements.
     + Visual compliance indicators to reflect regulatory mapping.
     + Integration of real-time collaboration features (e.g., co-editing status, version history).
     + Support for template application and customization.
3. **Document Management Interface:**
   * Build a Document Management component that:
     + Supports drag-and-drop file uploads with progress tracking.
     + Enables batch uploads, previews, and categorization.
     + Displays processing status and results (e.g., extracted text, compliance flags).
     + Provides version history and tagging for organization.
4. **Assessment and Grading Interface:**
   * Create an Assessment UI designed for instructors:
     + Implements one-click grading on a 1–4 scale.
     + Displays competency-based assessment forms.
     + Integrates digital signature capture and real-time performance trend visualization.
     + Supports offline assessment mode with auto-synchronization upon reconnection.
     + Allows feedback through speech-to-text integration and biometric data visualization.
5. **Analytics Dashboard:**
   * Develop an interactive Analytics Dashboard that:
     + Displays key performance indicators (KPIs) and training effectiveness metrics.
     + Offers drill-down capabilities on individual trainee performance and fleet-wide data.
     + Provides customizable widgets for real-time analytics.
     + Supports exportable reports (PDF, CSV, JSON) and integrates predictive insights.
     + Integrates visual elements like charts, graphs, and 3D data maps.
6. **Collaboration & Communication Components:**
   * Build UI components for real-time collaboration:
     + Smart workspace interfaces with co-editing, version history, and role-based access.
     + Integrated messaging modules (chat, notifications) for instructor–trainee communication.
     + Voice/video integration components with auto-transcription and meeting summary displays.
     + Contextual task management dashboards for converting AI insights into actionable tasks.
7. **Adaptive and Mobile Features:**
   * Develop adaptive UI elements:
     + Dynamic themes (dark mode, accessibility options such as dyslexia-friendly fonts).
     + Components that adjust based on user behavior and usage patterns.
   * Implement Progressive Web App (PWA) functionality:
     + Offline access via IndexedDB and service workers.
     + Background synchronization of training materials, assessments, and schedules.
     + Mobile-first design ensuring full compatibility with React Native (if needed) or responsive web views.
8. **API Integration & Services:**
   * Create robust API service layers to interface with the backend microservices:
     + Services for document upload/processing, syllabus generation, assessment data, scheduling, analytics, and collaboration.
     + Implement error handling, caching (via React Query), and real-time updates (WebSockets/Socket.IO).
     + Provide comprehensive API documentation within the code for maintainability.
9. **Testing, Performance, and CI/CD:**
   * Write unit tests (using Jest and React Testing Library) for each component.
   * Develop end-to-end tests (using Cypress) for critical user workflows (syllabus building, document upload, assessment grading).
   * Optimize for performance with code-splitting, lazy loading, and efficient state management.
   * Document CI/CD integration (using GitHub Actions) for automated testing and deployment to Vercel (or via Lovable).
10. **Documentation & Developer Guides:**
    * Include inline comments, a detailed README, and developer guides.
    * Generate style guides and component documentation (e.g., Storybook) for UI components.
    * Ensure thorough documentation of API endpoints and integration points with the backend.

**Final Note:**  
The frontend prompt must be detailed enough so that no functionality is skipped. It should cover every aspect from basic UI components through advanced 3D/AR visualizations, real-time collaboration, adaptive interfaces, and robust API integrations. The generated code should be complete, production-grade, and easy to integrate with the backend services to form a cohesive repository.

**All generated code must be structured according to this repository layout. Each piece of code must explicitly specify where it should be placed (e.g., whether it belongs in /components, /pages, /hooks, etc.).**