**Project Overview**

This web-based application is designed as an interactive school orientation quiz, modeled after the provided nursing simulation game. It simulates a guided learning experience where students input their personal details, answer multiple-choice questions about the school, receive immediate feedback on their choices, track their progress, and upon completion, receive a performance grade along with a downloadable PDF certificate. The certificate will include the student's full name, selected program, completion date, and grade.

The application will be built using HTML for structure, CSS for styling, and JavaScript for interactivity and logic. It will be responsive, accessible, and work offline via a service worker (optional but recommended for robustness). No backend is required, as all data is handled client-side. Questions will be stored in a JavaScript file for easy maintenance.

Key similarities to the attached simulation:

* Loading screen for initial setup.
* Start screen with input fields (name and program dropdown) instead of direct "Start Simulation".
* Progressive steps through questions (analogous to scenarios), with vital signs/v patient info replaced by question context if needed.
* Immediate feedback after each choice.
* Results screen with score, grade, statistics, and actions (e.g., download certificate).
* Modals for additional interactions (e.g., how it works, exit confirmation).
* Dynamic UI updates for progress, scores, and errors.

Assumptions:

* Questions are multiple-choice with one correct answer per question.
* Grading: Percentage-based (e.g., 90-100% = A, etc.), with a certificate generated using a library like jsPDF for PDF download.
* Programs in dropdown: Placeholder options (e.g., "Nursing", "Engineering"); can be customized.
* Date: Automatically uses the current date on completion.
* Assets: Folders for images and audio will be created, but content added later (e.g., background images, sound effects for feedback).

**Project Structure**

Organize the project in a root folder named "school-orientation-app". This ensures modularity and ease of maintenance. The structure will mirror the attached simulation for familiarity:

text

school-orientation-app/

├── index.html # Main HTML file containing all screens and modals

├── css/

│ └── styles.css # All CSS styles (responsive design, animations, themes)

├── js/

│ ├── app.js # Main application entry point; initializes everything

│ ├── questions.js # Data file containing quiz questions, answers, and feedback

│ ├── quiz-engine.js # Core logic for handling quiz flow, scoring, and feedback (adapted from game-engine.js)

│ └── ui-manager.js # Handles UI updates, modals, and DOM manipulations (adapted from ui-manager.js)

├── assets/

│ ├── favicon.ico # Site icon (placeholder; can be replaced)

│ ├── images/ # Folder for images (e.g., backgrounds, icons; empty initially)

│ └── audio/ # Folder for audio (e.g., success/failure sounds; empty initially)

└── sw.js # Service worker for offline support (optional)

* **Total files**: 8 (including folders).
* **Assets folders**: assets/images/ and assets/audio/ are created empty. Later additions could include:
  + Images: loading-spinner.png, background.jpg, certificate-template.png.
  + Audio: correct.mp3, incorrect.mp3, completion.mp3.

**File Breakdown**

1. **index.html**:
   * Defines the overall page structure using semantic HTML.
   * Includes sections for: loading screen, start screen (with name input and program dropdown), quiz screen (questions, choices, feedback), results screen (score, stats, certificate download), and modals (how it works, review answers, share/exit).
   * Links to CSS and JavaScript files.
   * Meta tags for viewport, description, keywords (e.g., "school orientation, quiz, education").
   * Accessibility features: ARIA labels, sr-only announcements.
2. **css/styles.css**:
   * Global styles: Reset, typography, colors (e.g., primary blue for buttons).
   * Component styles: Screens (flexbox/grid for layout), cards (patient info → question context), buttons, modals (overlay with animations).
   * Responsive design: Media queries for mobile/desktop.
   * Themes: Light/dark mode optional; loading spinner animation.
3. **js/questions.js**:
   * Exports an array of question objects.
   * Each object: { id: number, text: string, choices: array of strings, correctIndex: number, feedbackCorrect: string, feedbackIncorrect: string }.
   * Example: 10-15 questions about school policies, history, etc. (placeholders; user can populate).
4. **js/quiz-engine.js**:
   * Manages quiz state: Current question, score, errors, timer.
   * Functions: Load question, handle choice selection, calculate score/grade, generate certificate data.
   * Integrates jsPDF library (loaded via CDN in index.html) for PDF generation.
5. **js/ui-manager.js**:
   * DOM manipulation: Show/hide screens, update progress bar, render choices, display feedback.
   * Event listeners: Button clicks, form submissions (name/program).
   * Modal handling: Open/close for about, review, exit.
6. **js/app.js**:
   * Initializes the app on load: Hides loading screen, sets up event listeners.
   * Registers service worker.
   * Coordinates between quiz-engine and ui-manager.
7. **assets/favicon.ico**:
   * Basic icon for browser tab.
8. **sw.js**:
   * Caches static assets for offline use.

External dependencies:

* jsPDF (via CDN: <script src="https://cdnjs.cloudflare.com/ajax/libs/jspdf/2.5.1/jspdf.umd.min.js"></script>) for PDF certificate generation.

**Development Stages**

Develop the application in iterative stages to ensure systematic progress. Each stage builds on the previous, allowing for testing at milestones. Assume development uses a code editor (e.g., VS Code) and a local server (e.g., Live Server extension).

1. **Stage 1: Setup and Structure (1-2 hours)**
   * Create the project folder structure as outlined.
   * Write basic index.html skeleton: Include doctype, head (meta, links), body with divs for screens/modals.
   * Add empty css/styles.css and js files.
   * Test: Open index.html in browser; ensure no errors.
   * Goal: Static HTML layout visible.
2. **Stage 2: Styling (2-3 hours)**
   * Implement CSS based on the attached simulation: Use grid/flex for layouts, animations for loading/transitions.
   * Style screens: Loading (spinner), start (form inputs), quiz (cards for questions/choices), results (stats grid).
   * Ensure responsiveness: Test on different screen sizes.
   * Test: Visual consistency with attached HTML (e.g., feature-grid, btn classes).
   * Goal: Polished UI without functionality.
3. **Stage 3: Data and Quiz Logic (3-4 hours)**
   * Populate js/questions.js with sample questions (e.g., 5 for testing).
   * Build js/quiz-engine.js: State variables, functions for advancing questions, scoring (score = (correct / total) \* 100), grading (if-else for A-F).
   * Add certificate generation: Use jsPDF to create a PDF with text (name, program, date, grade).
   * Test: Console.log quiz flow; simulate answers.
   * Goal: Core logic works in isolation.
4. **Stage 4: UI Integration and Interactivity (4-5 hours)**
   * In js/ui-manager.js: Functions to render questions/choices, show feedback, update scores.
   * In js/app.js: Wire up events (e.g., start button submits form, stores name/program; choice clicks trigger feedback).
   * Handle start screen: Input validation (name required, program selected).
   * Add modals: How it works (static text), review answers (list questions with correct/incorrect), exit confirmation.
   * Implement download: On results screen, button triggers PDF download with filename like "orientation-certificate-[name].pdf".
   * Test: Full flow from start to end; check edge cases (all correct, all wrong).
   * Goal: Interactive app; user can complete quiz.
5. **Stage 5: Enhancements and Polish (2-3 hours)**
   * Add accessibility: ARIA, keyboard navigation.
   * Integrate assets: Placeholders in folders; reference in code (e.g., audio.play() on feedback).
   * Service worker: Cache files for offline.
   * Timer: Track time spent, display on results.
   * Error handling: Graceful failures (e.g., no name entered).
   * Test: Cross-browser (Chrome, Firefox), mobile, offline mode.
   * Goal: Production-ready app.
6. **Stage 6: Documentation and Handover (1 hour)**
   * Add comments in code for clarity.
   * README.md: Instructions to run (open index.html), customize questions, add assets.
   * Package: Zip the folder for sharing with the coding AI.
   * Goal: Ready for implementation or refinement.

**Implementation Notes**

* **Customization**: Questions in questions.js can be expanded. Program dropdown options in index.html (e.g., <select> with <option> tags). PDF Certificate: Simple layout – header "Certificate of Completion", body with details, footer with date. Security: No sensitive data; all client-side. Testing Tools: Browser dev tools for debugging; Lighthouse for performance/accessibility. Time Estimate: 13-18 hours total for a developer. Next Steps: Provide this plan to the coding AI, along with the attached documents for reference. The AI can generate the actual code files based on this blueprint. If needed, iterate by adding specific question examples or asset details. </option></select>