Presentation Outline

1. Intended User Description (30s)

- **Who is the intended user (based on application features)?**
  - The application is built for users needing to analyze patient data trends. Features support identifying high-risk groups (based on readmissions, conditions), analyzing demographics, understanding length of stay factors, and exploring specific patient records. This suggests users like healthcare analysts, researchers, or administrators.
- **Technical Challenges Addressed in Code:**
  - The codebase includes logic to parse potentially varied CSV formats (`parseCSVData`).
  - State management (`useState` , `useEffect` in React components) is used to handle data loading, filtering, and UI updates.
  - Error handling is implemented for data loading (`loadPatientData`), CSV parsing (`parseCSVData`), and API calls (`generateSummary` in `model/page.tsx`).
- **Solutions Implemented in Code:**
  - A detailed `parseCSVData` function handles type conversions, missing values, and potential formatting issues.
  - Client-side state management coordinates UI elements with data changes.
  - Error states are displayed to the user (e.g., using `AlertCircle` component).
  - Data caching (`cachedData` variable, `localStorage` usage) is used to optimize performance.

2. User Decision-Making Needs (1 min)

- **What decisions does the application support (based on features)?**
  - Identifying correlations between conditions and readmissions (`getReadmissionByCondition`).
  - Analyzing patient cohort demographics via age, gender, race, ethnicity distributions (`getDemographicDistribution` , `getAgeDistribution`).
  - Investigating patterns in length of stay (`getLengthOfStayDistribution`).
  - Tracking condition prevalence across age groups (`getConditionDistribution` , `getConditionsByAge`).

    - Generating high-level insights from data via AI summarization (`/api/summarize` integration in `model/page.tsx`).
    - Exploring individual patient details through filtering, searching, and pagination (`reports/page.tsx`).
- **How needs were addressed in the application:**
    - Specific data analysis functions were created in `lib/data-parser.ts` .
    - UI components in `reports/page.tsx` and `model/page.tsx` allow users to interact with these analyses (e.g., filtering, viewing summaries).
- **Technical Challenges Addressed in Code:**
    - Implementing multiple data aggregation and filtering functions (`get...` functions in `data-parser.ts`).
    - Handling asynchronous data loading and API calls (`useEffect` , `async/await`).
    - Managing UI state for filters, search terms, and pagination.
- **Solutions Implemented in Code:**
    - Modular functions in `data-parser.ts` perform specific analyses.
    - React hooks (`useState` , `useEffect`) manage component state and trigger data re-fetching/filtering.
    - API calls include error handling and loading state indicators.
- **Relevant Coursework:**
    - Mobile and Web Applications:

3. Data Validation and Preparation (1 min)

- **Process Implemented in Code:**
    - Data loading from user-uploaded CSV (`model/page.tsx`) or a default file (`/data/patient-data.csv`), with `localStorage` caching (`loadPatientData` in `data-parser.ts`).
    - CSV Parsing (`parseCSVData`): Splits text into lines, handles quoted fields, skips headers/empty lines.
    - Type Validation/Conversion: Converts strings to numbers (`parseInt` / `parseFloat` , checks `isNaN`), and to booleans (`parseBoolean`).
    - Handling Missing/Invalid Data: Uses fallback values (e.g., 0 for numbers) during parsing.
    - Performance Optimization: In-memory caching (`cachedData`) supplements `localStorage` .
    - Data Transformation: Calculates derived metrics and distributions (`get...` functions) for reporting.
- **Technical Challenges Addressed in Code:**

    - Robustness against different CSV structures/values is addressed via specific parsing logic in `parseCSVData`.
    - Client-side processing handles potentially large file parsing within the browser.
    - State management ensures data consistency across components.
- **Solutions Implemented in Code:**
    - The `parseCSVData` function includes checks for line length and data types.
    - Client-side parsing avoids server load.
    - `useState` manages data state.
    - Caching (`cachedData`, `localStorage`) improves responsiveness.
- **Relevant Coursework:**
    - Information Retrieval:

4. User Interface Design (1 min)

- **Design Implementation Highlights:**
    - UI built using `shadcn/ui` components (Cards, Tables, Buttons, Selects, Input) for consistency.
    - Application structure separates data ingestion/summary (`/model`) from detailed reporting (`/reports`).
    - Visual feedback provided via loading spinners (`LoadingSpinner`), success icons (`CheckCircle`), and error messages (`AlertCircle`).
    - Data interaction features: `CSVUploader`, preview table, download button (`handleDownload`).
    - Interactive reporting features: Paginated table, `SearchBar`, `Select` components for filtering, clear filter button.
    - AI Summary display: Uses a `Card` component with distinct loading/error/success states.
- **Technical Challenges Addressed in Code:**
    - Presentation of tabular data is handled using `shadcn/ui` `Table` component.
    - Filtering controls are implemented using controlled `Input` and `Select` components managed by `useState`.
    - Asynchronous operations (data loading, AI summary) are managed using `async/await` with UI state updates for feedback.
- **Solutions Implemented in Code:**
    - Leveraged `shadcn/ui` library for pre-built, styled components.
    - Used React state (`useState`) and effects (`useEffect`) to link UI controls to data filtering and display.

    - Implemented clear loading and error states in the JSX for asynchronous operations.
- **Relevant Coursework:**
    - Advanced Software Development & Software Development.

5. Final Reflection (3 min 30s)

- **Lessons Learned: Building Decision-Support Apps**
    - **Project Charter & Problem Definition:**
        - Clearly defining project goals and scope was crucial for focus and direction.
    - **Teamwork:**
        - Communication: Regular team meetings and open communication channels facilitated collaboration and issue resolution.
        - Coordination: Task assignment and tracking using project management tools ensured efficient workflow.
    - **Acquiring Domain Knowledge (Healthcare Analytics):**
        - Approach: Researching healthcare analytics concepts, metrics, and best practices through online resources and academic papers.
        - Resources/Tools: Utilized healthcare analytics libraries and frameworks (e.g., `d3.js` for data visualization).
        - What Worked/Didn't Work: Iterative design and testing with real-world data improved the application's effectiveness.
        - Discovery Process: Collaboration with team members from diverse backgrounds facilitated knowledge sharing and insight generation.
    - **Self-Efficacy (Individual Learnings):**
        - Learning New Concepts/Skills: Developed proficiency in Next.js, TypeScript, and data visualization libraries.
        - Working with Others: Improved negotiation and accountability skills through team collaboration.
        - Effort Estimation: Enhanced ability to estimate task complexity and duration through experience.
    - **Future Application:**
        - Trying to implement local llm based support for the application. As the outline for using with ollama is done.