**🏗 Architecture of the Application**

**1. Overview**

This is a **Markdown-based Slide Editor and Presentation Viewer**. It is a full-stack web application built using the **MERN-like architecture** (React frontend + Express backend), but with **SQLite** as the database. The application allows users to create, edit, view, and present slides with rich formatting and smooth animations.

**2. Client-Side (Frontend)**

**Technology**: React.js

* **Components**:
  + Home.js: Displays a list of all slides.
  + EditSlide.js: Markdown editor with live preview (uses Toast UI Editor or MDEditor).
  + SlideViewer.js: Displays one slide at a time with navigation, animations, and hotkeys.
  + Modal, Navbar, ProgressBar: Optional components to enhance UI/UX.
* **Markdown Rendering**:
  + Uses ReactMarkdown to render Markdown into styled HTML.
* **Styling**:
  + CSS modules or plain CSS for layout and transitions.
  + Animations done either using Framer Motion (optional) or pure CSS for slide transitions.
* **Routing**:
  + React Router handles navigation: / (home), /edit/:id, /new, /viewer.
* **Hotkey Support**:
  + Arrow keys: Next/Previous slide
  + ESC: Exit editing
  + Space: Toggle auto-play

**3. Server-Side (Backend)**

**Technology**: Node.js + Express.js

* **API Endpoints**:
  + GET /api/slides: Retrieve all slides
  + GET /api/slides/:id: Retrieve one slide
  + POST /api/slides: Create a new slide
  + PUT /api/slides/:id: Update an existing slide
  + DELETE /api/slides/:id: Delete a slide (optional)
* **Middleware**:
  + JSON body parsing (express.json())
* **CORS Enabled**:
  + Allows requests from frontend during development (localhost:3000 to localhost:5000)

**4. Database Layer**

**Technology**: SQLite

* Lightweight, file-based database ideal for small projects.
* Stores:
  + slides table with id, title, content, timestamp
* Queries handled via sqlite3 Node package.

**5. Slide Features**

* **Rich Text Editing**: Markdown editor with preview.
* **Animated Slide Show**: CSS transitions or optional Framer Motion.
* **Progress Bar**: Shows progress through slides.
* **Auto-play**: Automatically moves to the next slide.
* **Responsive Design**: Usable on desktop and mobile screens.
* **Keyboard Navigation**: Intuitive shortcuts for control.

**6. Project Folder Structure**

java

CopyEdit

project-root/

├── backend/

│ ├── index.js // Express server

│ ├── db.js // SQLite database setup

│ └── routes/ // API endpoints (optional)

├── frontend/

│ ├── public/

│ └── src/

│ ├── components/ // React components (SlideViewer, EditSlide, etc.)

│ ├── App.js

│ ├── index.js

│ └── styles/

├── package.json (x2) // For frontend & backend

**7. Deployment (optional for future)**

* **Frontend**: Deployed with Vercel, Netlify, or GitHub Pages
* **Backend**: Hosted on Render, Railway, or simple VPS
* **Database**: SQLite stored on server disk or upgraded to PostgreSQL for scalability

Let me know if you want this turned into a PDF or presentation slide format!

**🧠 Design Considerations**

**1. Technology Choices**

* **React for Frontend**: Chosen for component-based structure, ease of state management, and ability to integrate libraries like ReactMarkdown and Toast UI.
* **Express for Backend**: Lightweight, flexible, and fast for building RESTful APIs.
* **SQLite for Database**: Simple and file-based, ideal for prototyping and local storage without needing external hosting.

✅ *Consideration*: Easy to replace with PostgreSQL or MongoDB in the future if scalability becomes necessary.

**2. Markdown Support**

* **Why Markdown?**  
  It offers a lightweight way to format slides while keeping content editable and developer-friendly.

✅ *Future Feature*: Could support exporting to PDF or HTML from Markdown.

**3. Component Modularity**

* Designed components like EditSlide, SlideViewer, and ProgressBar as self-contained and reusable.

✅ *Future Feature*: Easily add themes, transitions, or plugin components like charts or quizzes.

**4. User Experience (UX)**

* **Animation Decisions**: CSS-based animations were chosen for performance and flexibility over JS libraries like Framer Motion.
* **Navigation Enhancements**:
  + Hotkeys for navigation and control.
  + Progress bar to visually track progress.
  + Auto-play toggle with spacebar.

✅ *Future Feature*: Add presentation timer, speaker notes, or multi-user collaboration.

**5. Extensibility**

* Routes and API design follow REST principles.
* Data model (title, content) is simple but can be expanded to include:
  + Tags
  + Authors
  + Slide templates
  + Version history

✅ *Future Feature*: Real-time syncing via sockets or Firebase.

**6. Styling and Responsiveness**

* CSS layout carefully crafted for mobile and desktop.
* Buttons are positioned smartly: navigation at edges, actions like edit/add in intuitive locations.

✅ *Future Feature*: Add responsive themes or a slide designer UI.

**7. Security & Performance**

* While not critical now, CORS and input validation are handled lightly.
* In production, future improvements may include:
  + Authentication (OAuth or JWT)
  + Rate limiting
  + Caching slide data

Would you like a concise version for use in a slide or interview answer?

- Any challenges faced or takeaways

*At first, it was really challenging because I couldn’t use UI libraries like Bootstrap or Material UI. I had to style everything manually using plain CSS. It was difficult, especially making the design responsive and visually appealing. But through this, I pushed myself to understand core CSS concepts better — like layout, animations, and transitions. In the end, I realized that knowing how to build styles from scratch is much more valuable than always relying on libraries. This experience gave me more confidence in writing clean, custom CSS and made me a better front-end developer.*