TASK FOR DAY 002

PLANNING THE TECHNICAL FOUNDATION.

1. Define Technical Requirements .

Technical requirements outline the technical aspects and specifications needed to achieve a project's goals. They are precise, measurable, and directly related to the technical implementation of the solution.

.Frontend Requirements:

These specify what the frontend must do to meet user and business needs.

.User Interface (UI):

.Design must follow a responsive layout, adapting to different screen sizes (mobile, tablet, desktop).
.Include intuitive navigation (e.g., a hamburger menu for mobile devices).
.Consistent design elements based on the style guide or design system.

.User Interaction (UX):

.Provide real-time form validation for inputs (e.g., email and passwords).
.Allow drag-and-drop functionality where applicable (e.g., uploading files).
.Support keyboard accessibility and ARIA roles for accessibility.

.Core Features:

.Authentication (e.g., login, sign-up, forgot password).
.Dynamic content rendering based on user data (e.g., user profiles or dashboards).

.Integration with APIs for real-time updates (e.g., fetching or posting data).

.Sanity CMS as Backend:

Frontend Requirements for Using Sanity CMS as Backend.

Sanity CMS is a highly customizable content management system that provides a headless architecture, allowing you to integrate it seamlessly with your frontend. Below are the key requirements and considerations for building a frontend using Sanity CMS as the backend.

1. Functional Requirements:

.Dynamic Content Fetching:

.Fetch data from Sanity CMS using GROQ (Graph-Relational Object Queries) or GraphQL.
.Implement queries to retrieve content for pages, sections, and reusable components.

.Content Rendering:

.Render dynamic content (e.g., blog posts, product listings) based on structured data from Sanity CMS.
.Support for real-time content updates using Sanity's real-time preview API.

.Routing:

.Generate dynamic routes for pages (e.g., `/blog/:slug` or `/product/:id`) based on content stored in Sanity.

.Content Previews:

Integrate Sanity's preview feature to allow editors to see content updates live before publishing.

.Image Handling:
.Use the Sanity Image API for responsive image rendering, transformations, and optimizations (e.g., cropping, resizing, and format conversions).

.Search Functionality:
.Implement search using Sanity queries to enable users to find specific content.

.User Authentication (if applicable):

Restrict access to specific pages or content sections by integrating authentication with the frontend.

2. Non-Functional Requirements:

.Performance:
. Optimize data fetching with GROQ queries by selecting only the required fields.
.Use caching strategies (e.g., ISR in Next.js or SWR for client-side rendering).

.Scalability:
.Ensure the frontend can handle large volumes of content by paginating data (e.g., blog lists or product catalogs).
.Build reusable components that map to Sanity's schema for future extensibility.

.Accessibility:
.Follow accessibility standards (WCAG 2.1) for rendering content dynamically fetched from Sanity.

.Localization and Internationalization:
.Leverage Sanity's built-in localization features for managing multilingual content.

- Integrate libraries like `next-i18next` for frontend language switching.

3. Technical Specifications:

.Technology Stack:

.Framework: Next.js (ideal for server-side rendering and static generation).

.State Management: React Context API or Zustand for local state.

.Sanity Integration:
. Install the official `@sanity/client` package for connecting to the Sanity backend.
Configure API credentials (project ID, dataset, and token) securely.

4. Integration Requirements:
.Real-Time Preview:
Implement preview mode using Sanity's webhook or real-time updates with `@sanity/client`.

.Third-Party Services:
Integrate analytics tools (e.g., Google Analytics or Mixpanel) to track user behavior on content pages.

.Deployment:
.Deploy on platforms like Vercel, Netlify, or AWS with environment variables securely stored.

.Error Handling:

Implement fallback UI for cases when Sanity content is unavailable or queries fail.

5. Developer Tools and Best Practices:

.Sanity Studio:

Customize schemas in Sanity Studio to define the structure of content displayed on the frontend.

.Environment Configuration:

.Store sensitive API keys and tokens in environment variables (`.env.local`).

.Version Control:

.Use Git for tracking changes, and maintain separate branches for staging and production environments.