Next.js Blog Post Application Requirements

Objective

To develop a **basic blog post application** using **Next.js 15** that showcases all major features of the framework, including the **App Router**, **server components**, **client components**, **dynamic routing**, **API routes**, **authentication**, **SEO optimization**, **image optimization**, **static generation**, **server-side rendering**, and more. The app will serve as a learning and demo tool.

Functional Requirements

1. Homepage

- Lists latest blog posts with title, excerpt, author, date, and thumbnail.
- Uses `fetch()` from a server component.
- Paginated or infinite scrolling.

2. Blog Post Page

- Dynamic route: `/blog/[slug]`
- Displays full blog content.
- Rendered statically (SSG) if possible or SSR otherwise.
- Includes metadata (title, og tags) using `generateMetadata`.

3. Create/Edit Blog Post Page

- Protected via authentication (admin-only).
- Rich text editor (Markdown or WYSIWYG).
- Uses client component + API call to submit.

- Form validation with Zod or Yup.

4. Authentication

- Uses `next-auth` (auth.js).
- GitHub, Google OAuth login.
- Session-based access control.
- Logged-in users can like posts, add comments.

5. Comments Section

- Comment form (client component).
- Displays threaded comments (SSR or ISR).
- Likes or reactions on comments (optional).

6. Admin Dashboard

- Route: `/admin`
- List, edit, delete blog posts.
- Add new categories/tags.
- Protected route using middleware and session checking.

7. Search and Filtering

- Client-side filtering by tag/category.
- Full-text search using server-side route or client search.

8. Tag and Category Pages

- Dynamic routes: `/tag/[tag]`, `/category/[category]`
- Lists all posts under a tag or category.
- Uses `generateStaticParams` and `generateMetadata`.

Technical Requirements

Next.js Features to Demonstrate

Feature	Description
App Router	Use `/app` directory for routing
Server Components	Fetch blog data from server components
Client Components	Rich text editor, search bar, comment form
Dynamic Routing	Blog posts, categories, tags
`generateMetadata()`	Dynamic SEO on each page
API Routes (`app/api`)	For CRUD operations on posts/comments

Feature	Description
Authentication (`next-auth`)	Secure login, protected routes
Layouts & Templates	Global layout, nested layouts
Static Generation (SSG)	For most blog and category pages
Server-Side Rendering (SSR)	For search and user-specific pages
Image Optimization	Blog post thumbnails with `<Image>`
Middleware	Session handling and route protection
ISR (Incremental Static Regeneration)	Revalidate content after changes
Error & Loading UI	`error.js`, `loading.js` per route
Environment Variables	For secure API keys
SEO/OG Tags	Using dynamic metadata
Deployment	Vercel or Docker deployment

Data Models

- **Blog Post**
- **Comment**

Blog Post

```
{
  id: string;
  title: string;
  slug: string;
  content: string;
  thumbnailUrl: string;
  authorId: string;
  createdAt: Date;
  updatedAt: Date;
  tags: string[];
  category: string;
}
```

Comment

```
{
  id: string;
  postId: string;
  author: string;
  message: string;
```

```
  parentId?: string;
  createdAt: Date;
}
```

Non-Functional Requirements

- **Responsive Design**: Mobile-first using Tailwind CSS.
- **Accessibility**: Authentication and Authorization.
- **Performance**: Lazy loading images, server components.
- **Security**: Auth, sanitization of user input.
- **Code Quality**: TypeScript, ESLint, Prettier.
- **Testing**: Unit tests (Jest), E2E (Playwright or Cypress).

Optional Enhancements

- Dark Mode toggle.
- Markdown editor with preview.
- Author profiles and bios.
- RSS Feed and sitemap generation.
- Email notifications on comments.
- PWA support.

Tech Stack

- **Frontend**: Next.js 15 (App Router), TypeScript, Tailwind CSS
- **Backend**: Next.js API routes or external backend
- **Database**: SQLite, PostgreSQL, or MongoDB (via Prisma or Mongoose)
- **Auth**: `next-auth`
- **Deployment**: Vercel / Docker + Render / Railway
