ChatGPT

Project Overview and Architecture

The **AppliedAgentic** website will be a modern, content-driven platform for publishing AI-related articles and course modules. It will use **Next.js** (latest version) for the frontend and backend (API routes), with data stored in a MySQL database. A custom admin panel (also in Next.js) will allow an administrator to log in (email/password only) and create, edit, and publish articles. All content – including module topics and individual articles – will be managed from the backend. Content editing will be rich-text (WYSIWYG) with support for images, embedded links, custom HTML, and formatting. The site will also integrate **OpenAI's SDK**: within the article editor, the admin can generate AI content (text via ChatGPT, images via DALL·E/GPT-Image, and audio via OpenAI's TTS) using API calls (with placeholders for API keys, model selection, temperature, etc.). The design will be **mobile-first and fully responsive** [1], with a modern, attractive UI (dynamic menus, smooth scrolling, and animations). For example, animations and transitions can be implemented using **Framer Motion**, a React animation library well-suited for Next.js that provides "smooth animations with simple props" and scroll/gesture-based effects [2]. The site's appearance should be clean and readable (good fonts, contrast, color) and follow best practices (minimalism, accessibility) [1].

Technology Stack

- **Next.js (React)** – A full-stack React framework for building server-rendered apps. Next.js supports API routes that can connect to databases. For example, Next.js API routes can use the `mysql2` library to connect to MySQL [3], or use an ORM like Sequelize or Prisma to manage data. Using Sequelize with MySQL allows automatic model-to-table synchronization [4]. Next.js is ideal for this project: it excels at performant, dynamic UIs and is proven in admin dashboards [5].
- **MySQL Database** – A relational database to store modules, topics, articles, users, etc. The schema will include tables for topics/modules and articles (with fields like title, content, status). Using an ORM like Prisma or Sequelize simplifies schema creation; e.g., Sequelize "supports model synchronization to automatically generate database tables and columns based on models defined in code" [4].
- **Hostinger Node Hosting** – Hostinger's managed Node.js hosting will be used (it explicitly supports React/Next.js deployments) [6]. Hostinger's platform auto-detects frameworks (including Next.js) and handles builds/SSL/CDN for you [6]. We can push the code via Git (GitHub integration) for instant deployment. Hostinger also offers managed MySQL databases.
- **Admin Authentication** – Authentication will be built in Next.js with email/password login. A common approach is NextAuth.js, which "abstracts away much of the complexity involved in managing sessions, sign-in and sign-out" in Next.js apps [7]. NextAuth can be configured with a credentials provider and MySQL (via Prisma) to allow admin logins. Alternatively, a custom API route using Next.js (like `/api/auth`) can handle login, hashing passwords (bcryptjs) and issuing JWTs or sessions [8] [7]. For simplicity and security, NextAuth (with a single admin user) is recommended.

Content Management & Data Model

- **Modular Content Structure** – The site will organize content hierarchically: **Modules → Topics → Articles**. For example, modules like "Foundations of Generative AI" will contain multiple topics (articles) listed under it. Each topic/article is a database record. This resembles a typical blog/

CMS structure. The MySQL schema might include tables such as `modules` (id, name, description), `articles` (id, module_id, title, content, status, created_at), and `users` (id, email, password_hash). Relationships ensure each article links to its parent module. Using an ORM (like Sequelize or Prisma) ensures referential integrity and makes queries easy [4].

- **Admin Panel** – A protected admin dashboard (e.g. at `/admin`) will let the admin manage modules and articles. This interface, built with React/Next.js components (possibly using a UI library like Material-UI or Ant Design), includes:
- **Topic/Article Editor**: A WYSIWYG rich-text editor for writing content. We can use a React-based editor (e.g. **React Quill** or **TinyMCE React**) inside Next.js. Such editors allow formatting, embedding images/links, etc. For instance, an example React Quill setup includes toolbar options like header, bold/italic, color, lists, *image* and *link* buttons [9]. In fact, one guide notes: *"a basic yet smooth and functional WYSIWYG editor using react-quill-new in Next.js"* [10]. TinyMCE (with its React integration) is another option and offers out-of-the-box image handling [11]. The editor will support uploading images: on image upload, the file will be stored (e.g. in `/public/uploads` or cloud storage) and inserted into the article content. Similarly, the editor will accept embed links or code as needed.
- **AI Content Tools**: Inside the editor, provide buttons or widgets to generate AI content. For text, the admin can click a "Generate Text" button that calls the OpenAI Chat API (e.g. GPT-4) with the current prompt; similarly an "Image" button can call the DALL·E/GPT-Image endpoint. OpenAI's Image API supports generation from prompts (it "lets you generate and edit images from text prompts, using GPT Image or DALL·E models" [12]). For audio, use OpenAI's Audio (TTS) API: its GPT-4o-mini TTS model can narrate text into speech (with 11 voices) [13]. The API key, model names, temperature, etc. will be configurable in the admin panel (stored in environment or admin settings). All AI API calls go server-side (Next.js API routes) so the key stays secret. Placeholders in the prompt (like `OPENAI_API_KEY`, `MODEL`, `TEMPERATURE`) will be used.
- **Content Status & Analytics**: The admin panel will list all articles with their status (published/draft). An admin can toggle visibility or publish immediately. For analytics, we can integrate simple view counters: e.g. a Next.js API route that increments a view count in MySQL when an article is accessed. Additionally, Next.js's built-in web vitals (via `useReportWebVitals`) or Google Analytics can track performance and pageviews [14]. For example, Next.js docs note it has "built-in support for measuring and reporting performance metrics" via `useReportWebVitals` [14]. The admin UI can then display basic stats (views, likes if added) per article if desired.

Editor Features and Media Handling

- **Rich Text Editing (WYSIWYG)** – As noted, we will embed a React rich-text editor. This must allow text formatting (headings, lists, bold/italic), inserting images and links, and even raw HTML if needed. React-Quill (a Quill-based editor) or Slate/Tiptap are popular choices. For example, a React-Quill toolbar can include `["link", "image"]` buttons to insert these elements [9]. TinyMCE's React integration is also robust and specifically designed for rich content (its blog recommends it as "the world's most trusted WYSIWYG HTML editor" with a React component [11]).
- **Image Upload** – The editor will support uploading images. On clicking "Insert Image," the admin can upload from disk. The app routes this to a Next.js API (e.g. `/api/upload`) that saves the image (local or S3) and returns a URL. The editor then inserts an `<img>` tag with that URL. The TinyMCE documentation on React image upload outlines connecting a rich-text editor to a server to handle file transfers [15], and React-Quill examples show similar flows. We will ensure file types are validated and storage is secure.
- **Embed Links & Media** – Similarly, embed links (e.g. YouTube iframes) can be pasted or inserted, and the editor's HTML mode will preserve these. We can allow raw HTML pasting if advanced formatting is needed.

- **Placeholders for OpenAI** – Within the editor interface, have a small form or side panel for AI generation. The admin can enter a prompt or select existing text, then click “Generate.” The client calls a Next.js API (using OpenAI’s Node SDK) to generate content. The response (text, image link, or audio URL) is then inserted into the editor at the cursor. This inline AI-assisted writing improves efficiency. All API parameters (API key, model choice, temperature) will be configurable (stored in server-side config or environment, not hardcoded).

UI/UX and Design

- **Mobile-First & Responsive** – The layout and styles will be responsive from the start. We will use CSS frameworks (or Tailwind) to create fluid grids and flexible images. Following best practices, we design for mobile first (the narrowest viewport), then add breakpoints for tablet/desktop. Media queries will adjust column layouts on smaller screens. A responsive approach ensures content reflows (as in fluid grids and images) [1] . For example, navigation menus collapse into mobile-friendly drawers.
- **Layout and Typography** – The site will have a clean, modern look. Good readability is key: use a sans-serif font optimized for the web, ensure high contrast, and adequate spacing. The design should not be cluttered – minimalism is advisable (the Interaction Design Foundation highlights that responsive designs should aim for minimalism and prioritize content [1]). Headlines and body text will follow a clear hierarchy. The example site “bytebytego.com” has a modern blog style, which we can emulate in spirit.
- **Dynamic Navigation** – The main menu will list modules (e.g., Module 1, Module 2) as sections. Hovering or clicking a module expands to show its topics/articles. This can be done with a React menu component and dynamic data (reading module names from the database). Animations (e.g. dropdown expansion) can be powered by Framer Motion for smooth effects.
- **Animations and Visual Flair** – For engaging UI, we’ll add subtle animations. Page transitions (e.g. fade or slide when navigating) can be implemented with Framer Motion’s `<AnimatePresence>` [2] . Scroll-triggered animations (elements fading in or moving into view) also use Framer Motion’s `whileInView` or similar. The guide notes Framer Motion supports “scroll-based animations for parallax and reveal-on-scroll” [16] . We may include a loading animation: perhaps a 3D spinner or logo animation (using CSS 3D transforms or a simple Three.js scene). Even a canvas-based 3D scene can show a loading progress bar for initial content.
- **Interactive Feedback** – Buttons and links will have hover/tap effects (scale up, color shifts) via Framer Motion’s `whileHover` props [17] . This makes the UI feel responsive.
- **Attractive Color Scheme** – Use a consistent color palette (e.g., from Material Design or similar) that’s vibrant but professional. Colors should not harm readability. Animations and graphics can add color (e.g., colored progress bars, accent elements).

Integration with Hostinger

Hostinger’s Node.js hosting supports Next.js out of the box [6] . We will set up a Hostinger Business or Cloud plan that allows: - **Deployment**: Connect the GitHub repo or upload a ZIP. Hostinger auto-detects Next.js and runs the build.
- **Database**: Use Hostinger’s managed MySQL database (set connection string in `.env`).
- **Environment Variables**: Store secrets (database credentials, OpenAI API key, NextAuth secret) securely via Hostinger’s interface.
- **SSL and CDN**: Enable free SSL certificates and CDN for fast global access.
- **Monitoring**: Use Hostinger’s dashboard for logs and scaling if needed.

Analytics and Monitoring

To **track article performance**, we'll implement: - **Next.js Web Vitals**: Use `useReportWebVitals` hook to capture metrics (FCP, LCP, etc.). Next.js docs note this hook for reporting [14]. These metrics can be sent to an analytics service or logged for monitoring.
- **Page Views**: Integrate Google Analytics or Plausible to count visits. Alternatively, custom counters in the database (increment on page load) can track reads. The admin panel can show simple stats per article (e.g. "Viewed 123 times").
- **Search Engine Optimization**: Ensure pages are SEO-friendly (Next.js SSR or static export for articles). Set proper `<meta>` tags for titles, descriptions. The responsive design will help SEO (mobile-first is a ranking factor [1]).
- **Error/Error Alerts**: Implement basic error handling (e.g. toast messages on save error) and optionally integrate Sentry or a similar service for crash reporting.

Summary of Components

1. **Database Layer (MySQL)** – Tables for Modules, Articles, Users. Use an ORM (Prisma/Sequelize) to define models and sync schema [4].
2. **API Routes (Next.js)** – Endpoints for CRUD operations on articles/modules (`/api/articles`, `/api/modules`), for file uploads (`/api/upload`), and for AI generation (`/api/ai/text`, `/api/ai/image`, `/api/ai/audio`). Use the OpenAI Node SDK in these routes (with placeholders for API key, model, etc.) [13] [12].
3. **Admin Frontend** – Protected pages under `/admin` or `/dashboard` for management. Includes a login page (NextAuth or custom), a dashboard listing articles, an editor page with the rich-text component, and settings page for API config. The layout uses a sidebar or top nav for navigation (like standard admin UIs).
4. **Public Frontend** – Article listing pages organized by topic/module, and individual article pages. These pages query the MySQL database (via API or getServerSideProps) to display content. They include share buttons, read time, etc. Content is styled for readability.
5. **AI Tools** – Custom buttons in the editor call our backend routes that invoke OpenAI's APIs. For example, "Generate Summary" could use a ChatGPT prompt, "Create Image" calls DALL·E (OpenAI's Image API) [12], and "Narrate" calls the Audio API for TTS [13]. The responses are inserted into the editor content.
6. **UI Enhancements** – Use **Framer Motion** for page transitions and element animations [2]. For instance, articles could fade in as the user scrolls, menus slide in, etc. A dynamic loading spinner (possibly animated in 3D) can show while fetching data.
7. **Styling** – Implement a consistent CSS (or Tailwind) theme. The design is inspired by modern tech blogs: clear typography, generous white space, and accent colors. Use high-quality fonts (e.g. Google Fonts like Inter or Roboto) for legibility.
8. **Mobile-First Layout** – Start with one-column mobile layouts: a collapsible menu (hamburger icon), stacked content. At breakpoints, switch to multi-column (e.g. sidebar next to content on wide screens). Use relative units and media queries as per responsive design best practices [1].

By combining these elements, the **AppliedAgentic** site will be a scalable, maintainable platform. The admin panel (built in Next.js) will give full control over topics and articles. The integrated AI tools (using OpenAI) will aid content creation. The mobile-friendly, animated frontend will deliver an engaging user experience [1] [2]. All code should be well-structured, with the database schema handled by ORM migration scripts, and deployment set up on Hostinger with the necessary environment configurations [6] [4]. The final product will resemble a modern tech-education site (akin to bytebytego.com) but fully custom-built as specified.

Sources: Next.js and MySQL integration [3] [4]; Hostinger Node hosting supports Next.js [6]; Responsive design best practices [1]; React rich-text editors (Quill/TinyMCE) usage [9] [11]; OpenAI API capabilities for image and audio generation [12] [13]; Next.js admin dashboards and auth (NextAuth) examples [5] [7]; Framer Motion for animations [2]; Next.js analytics (web vitals) [14].

---

[1] Responsive Design: Best Practices | IxDF
https://www.interaction-design.org/literature/article/responsive-design-let-the-device-do-the-work?srsltid=AfmBOoowX9wO-pPAH0im9xs9aMqEbQFJFhRA7FLuokB0SuAhcC69kHuJ

[2] [16] [17] A Beginner's Guide to Framer Motion in React & Next.js | by Ciril P Thomas | Medium
https://medium.com/@cirilptomass/a-beginners-guide-to-framer-motion-in-react-next-js-2378c7c1b20d

[3] Next.js API: Connect to MySQL Database | by Asish Panda | Medium
https://medium.com/@asishpanda444/next-js-api-connect-to-mysql-database-using-next-js-c020fa17a299

[4] Next.js 13 + MySQL - User Registration and Login Tutorial with Example App | Jason Watmore's Blog
https://jasonwatmore.com/next-js-13-mysql-user-registration-and-login-tutorial-with-example-app

[5] Step-by-Step Guide to Building an Admin Dashboard with Next.js - DEV Community
https://dev.to/hitesh_developer/step-by-step-guide-to-building-an-admin-dashboard-with-nextjs-26e4

[6] Node.js Hosting | High-Performance App Hosting
https://www.hostinger.com/web-apps-hosting

[7] App Router: Adding Authentication | Next.js
https://nextjs.org/learn/dashboard-app/adding-authentication

[8] Implementing Email & Password Authentication in Next.js App Router with Prisma and MySQL and bcryptjs for password hashing.* | by Amrita Vidhate | Medium
https://medium.com/@amritavidhate/implementing-email-password-authentication-in-next-js-7072a4a8e163

[9] [10] Rich text editor with React Quill in Next.js. - DEV Community
https://dev.to/kalama_ayubu_920a009aeba9/rich-text-editor-with-react-quill-in-nextjs-2o20

[11] [15] Mastering React image upload | TinyMCE
https://www.tiny.cloud/blog/react-image-upload/

[12] Image generation | OpenAI API
https://developers.openai.com/api/docs/guides/image-generation/

[13] Text to speech | OpenAI API
https://developers.openai.com/api/docs/guides/text-to-speech/

[14] Guides: Analytics | Next.js
https://nextjs.org/docs/pages/guides/analytics