ChatGPT

Specification for Multi-Domain Content Platform & Admin Dashboard

Overview

The goal is to design **five content domains**—Adult Health Nursing, Mental Health Nursing, Child Nursing, Social Work and Technology & Innovation (AI/Crypto)—within an existing React-19 + Vite + TypeScript + Tailwind/ShadCN application. These pages must support rich multi-media content (long articles, code blocks, audio, video, images), allow public reading, and offer engagement features for authenticated users. In addition, a **production-grade administrative dashboard** will provide full content management (CMS) capabilities, asset management, messaging and analytics. Implementation should integrate with the existing stack (Clerk for authentication, Cloudflare D1 for relational data, Cloudflare R2 for storage) and follow mobile-first design principles.

To give each domain its own identity, designers can use bespoke icons and illustrations that represent the subject matter. For example, the following composite shows conceptual icons for health care, mental well-being, child/family care and technology. These can inspire the hero images or accent graphics used on each page.

Why mobile-first matters

Research emphasises that mobile usage is ubiquitous—94 % of internet users accessed the web via smartphones in 2024 [1]. Mobile devices generated 77 % of global retail website traffic and 68 % of online orders [1]. Consequently, **mobile-first design** means starting with small screens and progressively

enhancing for larger viewports [2]. Wireframes and prototypes should be created for mobile first [2], with core functionality prioritised before adding progressive enhancements. Best practices include:

- Establishing a clear **visual hierarchy** by varying element sizes, contrast, typography and whitespace [3].
- Creating an **optimised UI/UX**: conventional placement of navigation, prominent clickable elements (minimum ~48 dp), and legible typography (14–19 pt) [4].
- Making controls **thumb-friendly**; research shows roughly half of users interact using one hand [5], so primary actions should sit within easy reach.
- Optimising media: images and graphics must load quickly (compress heavy assets by 70–90 %) [6].
- Following a systematic process: research, wireframing, prototyping, UX/UI design and testing [7].

What a modern CMS must offer

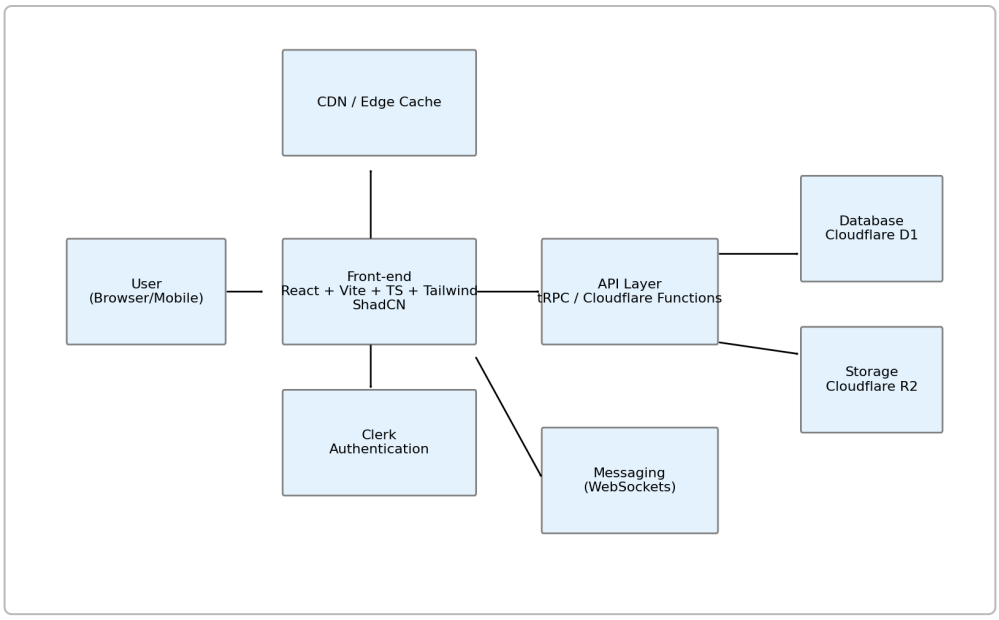
Modern content management systems are expected to simplify content creation while providing flexibility, security and scalability. Pipedrive's CMS feature guide lists several non-negotiable capabilities:

- **User-friendly interfaces**: WYSIWYG editors, drag-and-drop layout tools, clear menu structures, customizable dashboards and inline editing [8].
- **Customization & flexibility**: theme libraries, plugin ecosystems, API access, custom field definitions, modular architecture and multi-site capabilities [9].
- **SEO tools**: editable meta tags, URL structure control, XML sitemap generation, schema markup, mobile optimisation, performance tools and content analysis [10].
- **Multilingual support**: translation management, language-specific URL structures, automatic language detection and right-to-left (RTL) support [11].
- **Security**: regular security patches, two-factor authentication, SSL/TLS encryption, role-based access control, backup/restore and GDPR compliance [12].
- **Scalability & performance**: cloud hosting, caching, CDN integration, database optimisation, modularity and load balancing [13].
- **Integration**: robust API support and pre-built connectors to other services such as payments, email or analytics [14].

React-admin (an open-source framework used by thousands of companies) summarises the building blocks needed for a rich admin interface: responsive layouts, routing, real-time updates, caching, notifications, forms and validation, roles & permissions, search/filter, preferences, file upload, batch actions, import/export, history/versioning, theming and single-sign-on support [15]. These features inform the design of our administrative dashboard.

1. System Architecture

The platform follows a **client-server architecture** with serverless functions. The front-end is a single-page React application compiled with Vite and TypeScript; it uses ShadCN (Radix-UI-based components) and Tailwind CSS for styling. Authentication is handled by Clerk, and content/data is fetched through a tRPC (or REST) API layer deployed on Cloudflare Functions. Cloudflare D1 acts as the relational database for metadata (posts, comments, users, etc.) while Cloudflare R2 stores large assets (images, videos, audio). A WebSocket-based messaging service enables real-time chat. A CDN caches static assets.



2. Database Schema (Logical Data Model)

The following entities support the multi-domain content platform and admin dashboard:

Table	Purpose / Key fields
Domains	Defines each content domain (e.g., Adult Health, Mental Health, Child Nursing, Social Work, Technology). Fields: `id`, `slug`, `name`, `description`, `created_at`, `updated_at`.
Posts	Stores articles, research papers, code snippets and multimedia posts. Fields: `id`, `domain_id` (FK → Domains), `author_id` (FK → Users), `type` (text, code, video, audio, image), `title`, `slug`, `summary`, `content` (MDX), `status` (draft/published/scheduled), `published_at`, `created_at`, `updated_at`.
Attachments	Links files stored in R2 to posts or messages. Fields: `id`, `post_id` (nullable), `message_id` (nullable), `filename`, `url`, `type` (image/video/audio/document), `size`, `metadata`, `created_at`.
Users	Registered users authenticated via Clerk. Fields: `id`, `clerk_id`, `name`, `email`, `role` (admin/editor/user), `created_at`.
Comments	Stores nested comments on posts. Fields: `id`, `post_id`, `user_id`, `parent_comment_id` (self-reference), `content`, `created_at`, `updated_at`, `status` (approved/pending/flagged).
Reactions	Records likes/up-votes or other reactions on posts or comments. Fields: `id`, `user_id`, `post_id` / `comment_id`, `type` (like/clap/star), `created_at`.
Tags & PostTags	Tags categorise posts. `Tags` (id, name, slug), `PostTags` (post_id, tag_id).

Table	Purpose / Key fields
Messages	One-to-one or one-to-many messages between admins and users. Fields: `id`, `sender_id`, `receiver_id` (null for broadcast), `content` (MDX), `created_at`.
Notifications	System-generated messages for user engagement (new comment, reply, message). Fields: `id`, `user_id`, `type`, `data` (JSON), `read_at`, `created_at`.

Indices should be added on frequently queried fields (e.g., `slug`, `domain_id`, `post_id`, `user_id`), and foreign keys enforce referential integrity.

3. Front-End Implementation

3.1 File structure

The codebase should follow a modular structure for maintainability. One possible organisation:

```
src/
├─ components/           # shared UI components (Buttons, Cards, Modals,
Loaders, etc.)
├─ layouts/              # application and dashboard layouts
├─ pages/                # top-level route components per domain and admin
sections
│  ├─ domains/
│  │  ├─ adult-health/     # Adult Health domain page components
│  │  ├─ mental-health/
│  │  ├─ child-nursing/
│  │  ├─ social-work/
│  │  └─ technology/
│  ├─ posts/[slug].tsx    # dynamic post page (MDX rendering, comments,
engagement)
│  └─ admin/
│     ├─ dashboard.tsx    # admin home / analytics
│     ├─ posts/           # list, create, edit posts
│     ├─ files/           # file library
│     ├─ messages/        # messaging centre
│     ├─ users/           # user management
│     └─ settings/
├─ features/             # domain-specific hooks and slices (content fetching,
search, comments)
├─ hooks/                # reusable hooks (useAuth, useInfiniteScroll,
useDarkMode)
├─ services/             # API clients (tRPC/REST), R2 upload helpers
├─ contexts/             # React contexts (ThemeProvider, NotificationProvider)
├─ types/                # TypeScript interfaces and Zod schemas
├─ utils/                # utility functions (slugify, timeAgo, markdown
```

```
parser)
├─ routes.tsx            # defines routes with React Router v6
└─ main.tsx              # Vite entry point
```

3.2 MDX & Code Rendering

Posts are written in **MDX**, allowing Markdown syntax mixed with React components. The MDX parser (e.g., `@mdx-js/react`) converts Markdown to React elements. Code blocks use a high-performance syntax highlighter (e.g., `shiki` or `prism-react-renderer`) to render code similar to ChatGPT; MDX components can wrap code blocks for copy-to-clipboard functionality. Video and audio are embedded using `<video>` / `<audio>` tags with custom players (e.g., `react-player`) or Cloudflare Stream. Image optimisation can leverage `<next/image>` equivalent wrappers or Cloudflare image resizing.

3.3 Domain pages

Each domain page is a public landing page summarising recent posts, trending topics and curated resources. The following wireframe shows the general structure:

Key elements:

1. **Header/Nav** – global navigation, domain selector, search bar and quick links. On mobile, use a hamburger menu and bottom navigation for one-hand reachability [5].
2. **Hero section** – domain branding with background illustration (e.g., health-related vector) and a tagline explaining the scope. Use responsive typography and short call-to-action.
3. **Content feed** – paginated or infinite-scroll list of cards. Each card displays title, author, thumbnail, reading time, media indicator (video/audio/icon) and tags. Provide sorting (Latest, Trending, Featured) and filtering by media type, tags or publication date.
4. **Sidebar / Drawer** – shows trending posts, featured research papers, upcoming events or quick filters. On mobile, this becomes a collapsible drawer.
5. **Footer** – contains domain description, disclaimer and navigation to other domains.

3.4 Post page

The post page renders the full MDX content with interactive components:

- **Header**: title, author avatar (with link to profile), publication date, estimated reading time, domain badge and tags.
- **Content body**: MDX-rendered article. Use readable line length (~65–75 characters) and comfortable typography (16 px+). Code blocks have syntax highlighting and copy buttons; images have captions and lightbox; videos have a player; audio has a waveform player.
- **Engagement panel**: like/up-vote button, share menu (copy link, share to social), bookmark/ favourite. A reading progress bar shows how far the reader has scrolled.
- **Comments section**: nested comments with replies, markdown support and code formatting; sort by newest/oldest/best; load more on scroll. Only authenticated users can comment/like; guests are encouraged to sign in via Clerk.
- **Related posts**: suggests posts from the same domain or tags.

3.5 User dashboard (existing)

The existing user dashboard should expose bookmarks, personal reading history, saved searches and messaging inbox. Integrate notifications for replies and messages. Provide settings for dark mode, language and accessibility preferences.

4. Admin Dashboard Implementation

The administrative panel is only accessible to users with the `admin` or `editor` role. It uses the same React + Vite stack but has a distinct route namespace (`/admin`) and layout. The following wireframe illustrates the layout:

4.1 Route structure

```
/admin
├─ dashboard          # charts with engagement metrics (views, comments, likes)
per domain
├─ posts
│  ├─ list            # searchable list of posts with filters (status/domain/
tag)
│  ├─ create          # WYSIWYG/MDX editor with preview, scheduling and metadata
forms
│  └─ edit/:id        # edit existing post, show version history
├─ files
│  ├─ library         # browse/upload files stored in R2, preview thumbnails,
search by type/date
│  └─ upload          # drag-and-drop upload with progress bar and metadata
tagging
├─ messages
│  ├─ inbox           # list of conversations, unread counts, search
│  └─ conversation/:id # chat view (real-time via WebSocket), attachments
├─ users
│  ├─ list            # user table with search, filter by role, registration
date
│  └─ detail/:id      # user profile, posts/comments history, role assignment
├─ comments           # moderation queue for pending/flagged comments
└─ settings
   ├─ general         # site name, base URL, SEO meta tags, available domains
   ├─ roles           # define roles & permissions, assign editors to domains
   └─ integration     # API keys, CDN settings, email providers
```

4.2 Key features

1. **Content management** – WYSIWYG/MDX editor with support for code blocks, image/video upload (R2 integration), audio embedding, tables and diagrams. Use drag-and-drop to reorder content

blocks. Provide preview for different breakpoints (mobile/tablet/desktop). Implement scheduling with timezone selection and status workflow (draft → review → published). Maintain version history so editors can revert to previous revisions (React-admin emphasises history & versioning [15]).

2. **File library** – A digital asset manager built on R2. It allows admins to upload images, videos, audio or documents, generate thumbnails, view metadata (size, type, usage count) and assign assets to posts. Support folder/tag organisation, search and batch actions (delete, move). Provide direct links for external embedding.

3. **Messaging** – An internal messaging system enabling admins to communicate with authenticated users. Conversations support text, file attachments and code snippets (rendered as MDX). Use WebSocket channels for real-time updates and show typing indicators. Admins can broadcast announcements to all users or selected segments (e.g., domain subscribers). Each message thread persists in the `Messages` table with attachments in R2.

4. **User management** – List users with search and filters. Show registration date, role, activity (posts/comments). Allow admins to assign roles (admin/editor/user), suspend accounts or reset passwords via Clerk. Provide impersonation mode (log in as user) for support.

5. **Comment moderation** – Moderation queue lists pending or flagged comments. Admins can approve, edit, reply or delete comments. Provide filters (domain, user, post) and actions (bulk approve, mark as spam). Implement automatic spam detection using heuristics or third-party services.

6. **Analytics & Insights** – Dashboard displays charts summarising views, likes, comments, average reading time and engagement per domain. Use charting libraries (e.g., `react-chartjs-2` or `recharts`) to render bar/line/pie charts. Show top posts, growth over time and user demographics. Combine data from D1 and R2 usage stats. Provide CSV export. (React-admin lists charts, search/filter and notifications as foundational features [15].)

7. **Settings** – Control global site configuration: domain definitions, SEO meta tags, theme options (light/dark), supported languages, feature toggles. Manage API keys for third-party integrations (email, payment, analytics). Set default role for new users. Provide ability to update Terms of Service and Privacy Policy pages.

4.3 Roles & permissions

Role-based access control is crucial for security. At minimum define:

- **Admin** – full access to all domains, posts, users, messages and settings.
- **Editor** – can create, edit and publish posts within assigned domains; manage comments; limited access to analytics; cannot change global settings.
- **Contributor** – can create drafts but cannot publish; cannot access admin dashboard except their drafts.
- **User** – default role; can view public content and comment/like; cannot access admin areas.

Permissions should be enforced both in the UI (hide inaccessible navigation links) and in the API (verify JWT/role before performing actions). React-admin emphasises roles & permissions as a core feature [15]. Use Clerk's JWT token with custom claims to store role assignments.

5. Engagement & Interactive Features

Authenticated users should be able to engage with content to foster community and learning:

1. **Commenting system** – Supports nested replies, markdown formatting and code blocks. Comments display author avatar, timestamp and like button. Show the number of replies and collapse threads. Real-time updates when new replies are posted.

2. **Reactions** – Provide quick-reaction buttons (like/clap/star). Display reaction counts. Prevent duplicate reactions per user.

3. **Sharing & bookmarking** – Let users share posts via copy link, Twitter/X or LinkedIn; allow bookmarking posts to personal dashboard. Show a count of shares/bookmarks.

4. **Notifications** – Notify users when their post receives a comment, reply or message. Use Clerk's webhook or create a serverless event handler to push notifications via WebSocket or email.

5. **Search & filter** – Full-text search across titles and content. Filter by tags, domain, author, media type and date range. Provide auto-suggested search terms and highlight matched keywords. Implement with D1's full-text search or external search service.

6. **Accessibility & personalisation** – Offer dark/light modes, adjustable font sizes and high-contrast mode. Provide translation for UI labels (i18n). Persist preferences in local storage or user profile. Follow ARIA guidelines and ensure all interactive elements are keyboard-navigable.

6. Deployment & DevOps

- **Hosting** – Deploy the front-end on **Cloudflare Pages** for global edge delivery. Deploy API routes (tRPC/REST) as **Cloudflare Functions** or **Workers**, co-located near the database for low latency. Use Cloudflare Access to restrict admin routes.
- **Database** – Use **Cloudflare D1** with schema defined via migrations (e.g., `drizzle-orm`). Backup regularly and enable point-in-time recovery. Create read replicas for analytics.
- **Storage** – Store media in **Cloudflare R2**. Use pre-signed URLs for uploads and downloads; automatically generate thumbnails. Configure R2 bucket policies to allow public read for published assets while protecting private uploads.
- **CI/CD** – Configure GitHub Actions to run TypeScript type-checking, unit tests, end-to-end tests (Cypress/Playwright) and deploy on push to `main`. Use environment variables for API keys and secret tokens. Trigger preview deployments for feature branches.
- **Performance & SEO** – Use lazy loading, code-splitting and image optimisation. Add meta tags, Open Graph tags and JSON-LD schema for search engines. Pre-render important pages on the server or during build for improved SEO. Implement service workers for offline caching of content.

- **Monitoring** – Integrate logging and monitoring (e.g., Sentry for error tracking, Logflare for logs). Use analytics (e.g., Plausible or Google Analytics) to monitor traffic and engagement. Set up alerts for performance regressions or error spikes.

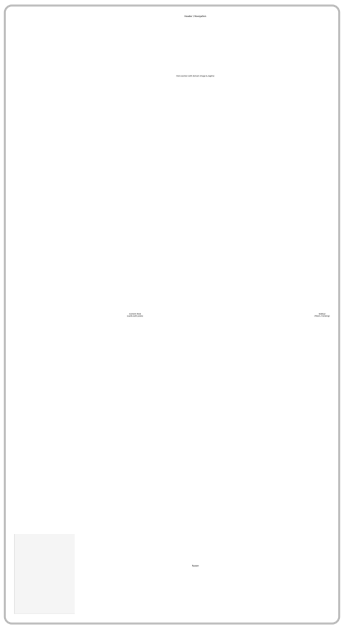
8. Future Enhancements

- **Multi-language content** – Add language selection and translation management to support global audiences [11].
- **AI-powered summarisation** – Offer automatic summaries or audio versions of long articles using speech synthesis.
- **Notifications digest** – Weekly/monthly email summarising new posts, trending topics and user statistics.
- **Gamification** – Earn points/badges for reading, commenting or contributing; display leaderboards to increase engagement.
- **API for third-party apps** – Expose a secure public API to allow mobile apps or partner sites to fetch content. Provide API keys with rate limiting and scopes.

9. Mockups

The following mockups illustrate the layouts described above. They are conceptual and focus on information architecture rather than final visual design. Colours and typography should be refined using Tailwind CSS and ShadCN components.

- **Domain page wireframe** – shows header, hero, content feed, sidebar and footer.



- **Admin dashboard wireframe** – shows side navigation, top bar and main content area where lists, editors and charts appear.

- **Technology & Innovation illustration** – abstract art representing AI and cryptocurrency; this can be used as a hero image for the Technology domain.

These mockups should be refined into high-fidelity designs using Figma or similar tools during implementation. They highlight the primary sections and emphasise mobile-first layouts (stacking columns, collapsible sidebars and comfortable tap targets). Designers should follow the visual hierarchy, thumb-friendly placement and media optimisation best practices noted earlier [4] [5].

---

1 2 3 4 5 6 7 Mobile First Design: Guide and Best Practices | Codica
https://www.codica.com/blog/mobile-first-design/

8 9 10 11 12 13 14 7 Must-Have Content Management System Features | Pipedrive
https://www.pipedrive.com/en/blog/content-management-system

15 React-Admin - The Open-Source Framework For B2B Apps
https://marmelab.com/react-admin/