ChatGPT

Deep Engineering Website UI/UX & Content Guide

Information Architecture

- **Homepage** – A full-screen hero section with a powerful tagline and background (image or subtle animation). Below the hero, include key highlights: an **Overview of KPP Technology**, a snapshot of **Projects** (e.g. planned 300 MW in Kurdistan Region [1]), a brief **About Us** summary, and a **Call-to-Action** (contact link or button). Each section should tease deeper content with links to the relevant pages.

- **About** – Company background and mission. Highlight that Deep Engineering is an *"Iraqi Renewable Energy Project Developer"* founded in 2019 in Erbil (branch in Basra) [2]. Emphasize core focus areas (renewable generation, smart-grid retrofits, project planning [3]) and credentials. Include company achievements (e.g. *"90 MW KPP project in Samawah and 300 MW in KRG in development"* [1]) and the exclusive partnership with Rosch Innovations (technology provider) [4]. This page can be sectioned into *Who We Are*, *Our Focus*, *Our Team* (with a grid of team member profiles), and *Our Partners*. Keep the tone professional and visionary.

- **Technology (KPP)** – This section contains multiple subpages detailing the Kinetic Power Plant technology:

- **KPP Overview** – Explain in accessible terms what the Kinetic Power Plant is and why it's innovative. For example: *"The Kinetic Power Plant (KPP) is an innovative power generation system that uses buoyancy and gravity instead of fuel, producing electricity 24/7 with no combustion and zero emissions"* [5]. Highlight that it provides continuous, clean power **anywhere** without reliance on weather or fossil fuel [6]. Mention it's developed by Rosch (Germany) with global demonstrations [7]. This page should also list key benefits (clean, continuous, decentralized, scalable – see UX/Copy below).
- **How KPP Works** – A deep-dive into the science and mechanics. Use diagrams or animations to illustrate the **physics principles** (Archimedes' buoyancy, gravity, etc.) and a step-by-step walkthrough of the energy generation cycle. Keep text sections bite-sized with clear headings (e.g. *"Buoyancy-Driven Motion"*, *"Energy Conversion Cycle"*). Consider embedding a short explanatory video or interactive animation demonstrating the **air injection engine** and moving floaters (the presentation had an "Air Driven Engine" video [8] and step-by-step visuals).
- **KPP Components** – Detail the major system components and design. Break out sub-sections or pages for **System Design & Components** (e.g. floaters, water tank, chain system, generator, compressor, control system). Each component can have a brief description and technical highlight. For example: *"500 kW Low-Speed Permanent Magnet Generator – runs at 375 RPM with ~95% efficiency, enabling direct grid connection without gearboxes"* [9]. Use images or icons for each part (a water drop icon for water tank, bolt icon for mechanical parts, etc.). If content is long, consider splitting into "Components (Part 1, Part 2)" pages as in the technical overview.
- **Performance & Specs** – Provide technical credibility with data. Include subsections on **Performance** (e.g. output 1 MW and 5 MW module options, availability 100%, load profiles) and **Environmental Impact** (zero emissions, minimal water use – *"<5% of water recirculation needs periodic top-up"*, etc.). If

available, present the **Water Consumption Analysis** (the tech overview analyzes water usage [10]) and maintenance expectations. Use charts or infographics for clarity.

All technology pages should have a consistent sub-navigation (or clearly accessible links) so users can easily explore the different aspects. Maintain an authoritative but approachable tone (explain technical terms in plain language where possible).

- **Projects** – Showcase Deep Engineering's current and planned projects. Use a portfolio grid or list format with an image, project name, size, and status for each. For example, cards for each planned power plant: *Zakho – 100 MW (Kurdistan, in progress)*, *Soran – 100 MW*, *Raparin – 50 MW*, *Garmian – 50 MW* [11]. Also include the 90 MW Samawah project with the Iraqi Ministry of Electricity [12]. Each project entry can link to a detail page or pop-up with more info (location, partners, timeline). Focus on the impact (e.g. "providing clean energy for X homes"). Use consistent imagery (site photos or renderings, or use a map graphic with pins for project sites).

- **Team** – Profile the leadership and key team members. A clean grid of headshots with names, titles, and a brief bio or expertise area. Since the company has ~35 staff across mechanical, electrical, SCADA, finance disciplines [13], highlight the breadth of expertise. This page can also emphasize the company culture or any career opportunities (if relevant). Keep the design minimalistic – simple cards or a list that expands on click for the bio.

- **Contact** – Provide an easy way for visitors to get in touch. Include a contact form (name, email, message fields) with a clear call-to-action like "Send Message" and a success confirmation. Also list the company's contact details: address of HQ in Erbil and branch in Basra, phone number, and email. Optionally, embed a map for the HQ location. Ensure the form is CMS-ready: the form submission can be handled via an API or service, but the content (address, emails) should be editable via the CMS. Keep this page straightforward, with a short inviting text (e.g. *"Have questions or project inquiries? We'd love to hear from you."*).

The navigation menu should include the main sections: **Home, About, Technology, Projects, Team, Contact**. The "Technology" item can have a dropdown for the subpages (Overview, How It Works, Components, etc.) or go to the overview with internal links. Use a sticky header so navigation is always accessible. Also include footer navigation with these links plus social media if applicable.

UX & Copywriting

The content should be **clear, engaging, and technically credible**. All copy should reflect a confident, innovative tone – positioning Deep Engineering as a leader in cutting-edge renewable energy in Iraq. Below are key content elements and suggested copy:

- **Hero Section (Homepage):**
- *Tagline:* A succinct, powerful statement encapsulating the value proposition. For example: **"Continuous Clean Energy, Anywhere."** This conveys 24/7 renewable power available at any location, echoing KPP's unique benefit of siting power plants wherever needed [6].
- *Subtagline:* One sentence for clarity. E.g.: *"Delivering 24/7 renewable power through the revolutionary Kinetic Power Plant (KPP) technology – no fuel, no emissions."* This reinforces the key differentiators (continuous, fuel-free, clean) [5].

- *Call to Action:* A prominent button like **"Learn More"** (scrolls to Technology section) or **"Our Technology"**, and maybe a secondary **"Contact Us"** button for immediate inquiries.

- **About Page Copy:** Use an **"About Us"** heading or a brief intro: *"Deep Engineering is Iraq's pioneer in renewable energy project development, turning innovative technology into sustainable power."* Expand on the company's story: e.g. *"Founded in 2019 in Erbil, with a branch in Basra, Deep Engineering has assembled a multidisciplinary team to drive Iraq's clean energy future* [2] [13]*."* Highlight credibility: *"As the exclusive KPP licensee for Iraq* [1]*, we are deploying world-class kinetic power plants to deliver reliable green electricity."* Keep paragraphs concise. Use subheadings for *Mission*, *History*, *Partners* etc., if needed. For instance, under *Our Mission*, a line like: *"To bridge the gap in reliable power supply by harnessing innovative technologies that provide clean, continuous energy for Iraq and beyond."*

- **Technology Section Copy:** This content must balance technical detail with clarity. Structure it with descriptive headings and bullet points:

- *Overview:* Provide a short intro as above. Then use a features list (with microcopy similar to Ki-Tech's messaging) to emphasize **Why KPP**:
    - **Clean Power:** *"Operates without any fuel, water, wind or solar input – generating electricity with zero emissions (no CO_2, NOx, or other pollutants)* [14]*."*
    - **Continuous Power:** *"Provides stable baseload power 24/7, independent of weather conditions* [15]*, with built-in redundancy ensuring near 100% uptime."*
    - **Decentralized:** *"Can be installed at the point of need – even off-grid locations – reducing transmission losses* [16]*."*
    - **Scalable:** *"Modular design allows capacity from a few megawatts to 100+ MW by paralleling units* [17] [18]*. Start with what you need and expand easily."*
    - **Small Footprint:** *"Requires far less land than solar or wind: ~300 m^2 per MW (vs. thousands for other sources)* [19]*."*
    - **Competitive:** *"Projected levelized cost ~€25/MWh* [20] *– delivering affordable energy along with grid stability benefits."*

      Each feature can have a heading (perhaps with an icon) and a one-sentence explanation, as above. This microcopy should be punchy and fact-based.
- *How It Works:* Explain in simple steps the **KPP cycle**. For example, a sequence might be: 1) *Air is injected into underwater floaters, causing them to rise.* 2) *The rising floaters drive a chain that turns a generator.* 3) *At the top, air is released and the floater sinks back down to repeat the cycle.* Use an active voice and present tense to make it engaging (like a tour of the process). Include a note on the physics: *"By leveraging Archimedes' Principle (buoyancy) and gravity in a closed-loop, KPP continuously converts mechanical motion into electricity."* Keep each step description short; consider using an infographic with labels.
- *Technical Details:* Write approachable descriptions for complex components. Instead of raw specs, frame them in terms of benefits: e.g., *"High-Efficiency Generator: A 500 kW permanent-magnet generator runs at low RPM, eliminating complex gearboxes and improving reliability* [9]*."* Or *"Smart Control System: An integrated SCADA ensures optimal performance and safety, automatically adjusting air input and monitoring output in real-time."* If there are unique terms (like SCADA), briefly explain ("Supervisory Control and Data Acquisition system"). Use metric units and standard terms consistently. Aim for a tone that is confident and factual, avoiding marketing exaggeration.

- Throughout the tech pages, use **microcopy** to guide the reader: e.g., labels on diagrams ("Buoyancy Drives Rotation → Power Generation"), captions under images ("KPP module schematic"), or tooltips for jargon. These small text elements should be concise and helpful. For example, on a diagram of the KPP module, micro-labels might point out *"Floaters rising in water (buoyant force)"*, *"Generator converts motion to electricity"*, etc.

- **Projects Page Copy:** For each project, include a short description that gives context. E.g., *"Zakho 100 MW – Four KPP units of 25 MW each, providing continuous power to the region. Partnered with KRG Ministry of Electricity, expected commissioning 2025."* Keep project descriptions 1-2 sentences, focusing on the outcome and status. An introduction paragraph for the page can state: *"Deep Engineering is currently developing 390 MW of KPP projects across Iraq* [1] *to deliver clean energy where it's needed most. Our projects range from large utility-scale plants to regional power solutions."*

- **Team Page Copy:** Open with a friendly invite like *"Meet the Team behind Deep Engineering"*. For each member, list their **name, title**, and a one-liner highlighting their expertise or role (e.g., *"John Doe – Project Lead, 10+ years in renewable project management"*). Microcopy tip: instead of a generic "Team" heading, consider something like **"Our Leadership & Experts"** for a bit more personality. If space allows, a quote or personal mission from the CEO/managing partner could be nice, but keep it short and relevant (e.g., *"Our goal is to secure a sustainable energy future for Iraq – Eng. Ranj Sherko, Managing Partner."*).

- **Contact Page Copy:** Keep it straightforward and welcoming. A heading like **"Get in Touch"** or **"Let's Talk"** works well. A sentence or two: *"Interested in our technology or projects? Send us a message and we'll respond promptly."* On the form, use microcopy to guide users: placeholder text like "Your Name", "Your Email", "How can we help you?" or field labels. The submit button could say **"Send Message"** and a confirmation message like *"Thank you for reaching out – we'll get back to you soon."* If listing addresses, you can use a subheading like **"Offices"** and list Erbil and Basra with full addresses. Ensure tone is polite and encouraging.

General Copy Style: Throughout the site, maintain a **professional and clear style**. Use short sentences and avoid overly technical jargon without explanation. Where technical terms are necessary, assume a smart but non-specialist reader and add a brief clarification (e.g., "500 kW (kilowatt) generator"). Write in active voice and present tense to make it immediate. For example, instead of "The KPP technology has been demonstrated to be effective," say *"KPP technology delivers reliable power – proven in demonstrations in Germany, Thailand, and more."* [7] . This keeps the text engaging.

Finally, ensure all headings and subheadings are informative. A user scanning the page should grasp the key points from headings alone. For instance, on the tech page, instead of a vague heading like "Innovation", use **"Fuel-Free, Emissions-Free"** or **"Continuous 24/7 Operation"** as section titles – this immediately conveys the message even without reading the paragraph.

Visual Design

The design will draw inspiration from Vestas's modern, minimalistic, and elegant aesthetic. We aim for a clean **corporate look with a renewable energy feel**. Key visual design guidelines:

- **Color Palette:** Base the palette on the Deep Engineering logo and Vestas-like blues. Likely a **deep blue** as the primary brand color (for example, a navy or azure tone similar to Vestas' blue). In the provided materials, a dark blue around `#18335A` was dominant in the logo [21]. Use this or a close hex (`#152D47` or `#04318F` as seen in the theme config [22] [23]) for headers, footer backgrounds, and call-to-action buttons. Complement it with a **bright accent blue** (e.g. `#2150FE` from the design theme [24]) for highlights, links, or hover states. White and light gray will provide the clean background and contrast. For instance, use an off-white (`#F0F0F1` or pure #FFFFFF) as the page background [25], with plenty of whitespace to echo a minimal feel. **Neutral grays** (e.g. #4C4C4D text gray [24]) should be used for body text and secondary elements, ensuring readability on light background. Consider a **secondary accent** in a warm tone (perhaps an orange or rust, e.g. `#C84209` was suggested in the style guide) used sparingly for things like important buttons or hover effects to add visual interest [22]. Overall, the palette should feel *"cool, professional, and modern"* with blues and grays [26], aligning with a clean energy theme.

- **Typography:** Use a combination of a professional sans-serif and a elegant serif, mirroring modern corporate trends. The provided theme uses **Crimson Pro** (serif) for headings and **Heebo** (sans-serif) for body text [27]. This is a strong pairing: Crimson Pro brings a touch of sophistication and trust (great for titles/taglines), while Heebo is clean and highly legible for content. Both fonts are Google Fonts, ensuring easy integration. If these exact fonts are not desired, alternatives could be **Merriweather or Playfair Display** for serif, and **Inter or Lato** for sans-serif – but sticking to Crimson Pro/Heebo is fine given they were already chosen. Font sizes should be generous, especially on the homepage – e.g. hero title ~4rem (scaling to maybe 6rem on large screens) to create an impactful statement, and body text around 1rem (16px) for comfortable reading. Use consistent line-heights (e.g. 1.5) and spacing. Headings should be bold or semi-bold (Crimson Pro 600 weight was used [28]) with a distinct color (maybe the deep blue) [23], whereas body text can be a dark gray for contrast against pure black (easier on eyes). Ensure the typographic hierarchy is clear: use different sizes/weights for H1, H2, H3, etc., and make sure link text or buttons stand out (perhaps in the accent blue).

- **Layout & Grid:** Use a **responsive grid system** with plenty of whitespace. Tailwind CSS's built-in grid and flex utilities will handle this. For a content width, consider using a max width container (~1280px or so) centered on larger screens (Tailwind's `container` class with appropriate breakpoints). Margins/padding should be generous to give an open, airy feel (inspired by Vestas' spacious layouts). For example, use a 12-column grid for the overall layout. The homepage sections can use full-width background with content centered in the max-width container. The "About" and "Technology" pages might use a two-column layout: text on one side, image on the other, on desktop (stacked on mobile). Maintain consistent gutters (space between columns, e.g. `gap-8` in Tailwind for 2rem gap). Use **fullscreen sections** where appropriate (the hero is fullscreen by design). For interior pages, a common approach is a hero banner area (shorter) with a page title, then content grid. Keep the visual hierarchy such that the most important content (headings, key images) draws attention first. Align text to a baseline grid and use Tailwind's utility classes to ensure consistency (e.g. same margin bottom on all headings, etc.). The design should be fully responsive: on mobile, use a single-column

layout, stacking elements; on desktop, introduce multi-column grids. Test for various screen sizes so that no text is too cramped or too wide (limit paragraphs to reasonable width for readability, ~65-75 characters per line is ideal).

- **Imagery & Illustration:** Favor **modern vector illustrations** and high-quality images that reinforce the content. Since KPP is a novel technology, consider using **3D renders or diagrams** to visually explain it (the technical overview included generated images of the KPP module, which can be repurposed). If available, incorporate imagery of kinetic power plant modules or concept art – perhaps stylized cutaways or schematic-like illustrations. The style prompt in the design file suggests *"Blues, grays, bright, professional, modern"* visuals [26] – so illustrations should use the brand colors (blue/gray) and a clean, flat or semi-flat style (think isometric industrial illustrations or outline icons with accent colors). Avoid cluttered or overly cartoonish graphics; everything should feel sleek and engineered. Icons will play a big role for quick visual communication: use **Heroicons** (a clean, consistent icon set) for things like feature bullets (e.g. a leaf icon for "Clean Power", a lightning bolt for "Continuous Power", a location marker for "Decentralized", etc.). Heroicons' outline style will match a minimalist aesthetic and can be styled with the brand colors easily. Use icons at a moderate size (24px or 32px) alongside text. All images and illustrations should be optimized for web (Next.js's Image component can help with responsive sizes).

- **Visual Consistency:** Maintain a consistent look across the site. This means using the same color shades for similar elements (Tailwind can have a custom theme configured with these brand colors, e.g., `primary: '#18335a'`, `secondary: '#2150FE'`, etc.). Buttons should have a uniform style (e.g. filled primary color for main CTAs, outline or ghost style for secondary actions). Form elements should use the same design language (e.g. light gray backgrounds, maybe slight border radius of 4px, blue focus outline for inputs to match brand color). Use subtle **shadows or depth** sparingly to highlight interactive cards or to make text on images readable. For example, project cards can have a small shadow (`shadow-md`) and hover elevation (`hover:shadow-lg`) to indicate interactivity, aligning with modern design trends. Animations (see next section) will also be part of the visual feel, but ensure they are subtle and do not detract from the clean design.

- **Accessibility & Contrast:** Given the professional nature, ensure sufficient contrast (the deep blue on white is good for text; avoid light gray text on white for body copy as it might be too low-contrast). Provide alt text for images (especially any diagrams explaining tech). Use accessible font sizes and consider color-blind friendliness (the blue/orange combo is generally okay, but check that any critical info isn't conveyed by color alone). A minimalistic design can still be very accessible by sticking to simple layouts and clear text.

In summary, the visual design should convey **innovation, trust, and cleanliness** – much like Vestas' site which uses large whitespace, clear blues, and modern typography to great effect. By using a coherent color palette (rooted in blues/whites), strong typographic hierarchy with a mix of serif/sans, and clean graphics, the Deep Engineering site will look both **professional and inviting** to stakeholders and visitors.

Animations

Animations will add a layer of modern interactivity, but they should remain **sleek and not overwhelming** (keeping with a minimal elegant feel). Here are recommendations for animation usage and libraries:

- **Library Choices:** We recommend using **Framer Motion** or **GSAP (GreenSock)** for implementing animations in Next.js. Both are well-suited for React/Next; **Framer Motion** integrates naturally with React components (declarative animations), whereas **GSAP** offers more fine-grained control for complex sequences. In general, Framer Motion shines for simple component animations and scroll-triggered reveal effects, while GSAP is powerful for timeline-controlled animations or advanced effects [29]. For most UI flourishes (fades, slides, modals), Framer Motion should suffice. If you plan on elaborate animations (like detailed SVG drawings or canvas effects), GSAP (possibly with its ScrollTrigger plugin) is a great addition. Both can coexist if needed. Additionally, consider using **React Spring** or **Framer Motion** for physics-based or interactive UI animations and **Lottie** (via `lottie-react` package) if you have any Lottie animation files (e.g., an animated diagram or logo motion graphic).

- **Hero Section Animation:** The homepage hero is a prime area for a subtle "wow" effect. For example, you could have a **background video or loop** (perhaps abstract water and air bubbles rising, hinting at KPP's principle) playing muted behind the tagline. If not a video, a **particle animation** or slowly animating graphic can work – e.g., floating geometric shapes or an outline of the KPP module slowly drawing in. Keep it low-key so text stays readable (use overlay or blur if needed). The hero text itself can animate in: for instance, using Framer Motion's `animate` prop to fade-up the tagline and subtext sequentially, with a slight delay, giving a polished entrance. A quick example: the tagline slides from the bottom with opacity 0 to 100% over 0.8s, and the subtagline fades in right after. The call-to-action button might have a **hover animation**, like a slight upward lift or a ripple effect on click (achievable with CSS or small JS). These animations draw attention without being distracting. Avoid heavy animations that could impact performance on load; use techniques like Next.js's lazy loading for any video or ensure the CSS animations are not janky.

- **Section Transitions:** As users scroll, each section (About, Technology, etc.) can have a gentle reveal. For example, the About section text might **fade in and rise slightly** as it enters the viewport (use Framer Motion's `whileInView` or a library like AOS – Animate on Scroll – if you prefer a utility approach). Images can have a slight **zoom-in** or **parallax drift** on scroll. A modern effect used in many renewable energy websites is a **parallax** background – e.g., an image of a wind farm (or in your case maybe a landscape with infrastructure) that moves slower than the foreground text when scrolling. You can implement this with CSS or JS (Framer Motion can animate `backgroundPosition` or use GSAP's ScrollTrigger to update on scroll). Ensure section transition animations are consistent: e.g., use the same duration and easing for all fade-ups (say 0.5s ease-out) so the site feels cohesive.

- **Interactive Elements:** Use micro-interactions to enhance UX. For instance, navigation menu items can have a **hover underline animation** (like a sliding underline or a color fade). Using Tailwind, you might do this with CSS transitions, or use a small Motion component to underline on hover. Buttons should have hover states (color change or slight scale-up) – Tailwind's `transition-transform` and `hover:scale-105` can do a quick scale, and `hover:bg-primary-darker` for color. Form

inputs might highlight on focus with a smooth box-shadow expansion (e.g., via CSS `transition` on focus). These subtle cues make the site feel modern and responsive to the user.

- **Scroll-based Effects:** Given the technical nature of KPP, consider **scroll-triggered animations to tell a story**. For example, on the "How KPP Works" page, as the user scrolls down a step-by-step diagram, you could animate each step: the floater rises, the chain moves, the generator spins – using an SVG or Lottie animation controlled by scroll position. This could be done with GSAP's ScrollTrigger, which can tie the progress of an animation to the scrollbar. Keep these animations lightweight; if Lottie, use the smallest JSON; if GSAP on images/SVG, ensure no huge DOM reflows. Another idea: animate numeric counters for stats (e.g., a counter from 0 to 300 MW when the Projects section comes into view, highlighting "300 MW under development"). Libraries like Framer Motion have `useAnimation` or one can do a simple incremental counter in React. This kind of animation adds excitement and draws attention to key metrics.

- **Performance & Preferences:** Always provide a **non-animated fallback**. If animations are disabled (prefers-reduced-motion media query), the site should still be fully usable. Design with performance in mind: use GPU-accelerated CSS transforms (translate, opacity) for smooth animations, and avoid heavy JS on main thread. Framer Motion and GSAP are both performance-optimized, but misuse can still lag, so test on mid-range devices. For Next.js specifically, animations should be implemented in client-side components (disable SSR for those if needed using dynamic import with `ssr:false` for heavy animated components to avoid server rendering issues).

- **Recommended Dependencies:**

- **Framer Motion:** Great for React-based animations; you can easily animate JSX elements. For example, animate presence for modals or page transitions. Next.js 13/15 can use the new App Router with React 18 – you might handle route transitions by wrapping `LayoutGroup` or use `AnimatePresence` on page changes for a smooth crossfade between pages. There are many examples of page transitions using Framer Motion in Next [30].
- **GSAP:** Excellent for complex timelines. If you want, say, a continuous floating animation of an object (like an SVG of a buoy bobbing up and down indefinitely) or fine control over scroll progress animations, GSAP is unrivaled. Use the official GSAP package; for ScrollTrigger, you'll need to import and register that plugin. GSAP can coexist with React – just be mindful of using `useEffect` to initiate GSAP animations on refs after mount.
- **Others:** *React Scroll Parallax* or **Locomotive Scroll** (for smooth scrolling effect) can add a polished feel if desired, but these are optional. A simpler approach is to use CSS `scroll-behavior: smooth` for anchor links (like clicking "Learn More" scrolls smoothly). If using heavy visual effects (like WebGL or 3D), look into **three.js** or smaller canvas libraries, though likely not needed unless you want a 3D model of KPP interactive on the site (advanced, can be skipped for now).

In implementation, start with small, meaningful animations – test them – and iterate. The goal is to **enhance the storytelling** (e.g., emphasize how KPP works or how professional the company is) without distracting or annoying the user. By using modern animation libraries and techniques carefully, the site will feel dynamic and cutting-edge, aligning with the innovative nature of KPP technology.

Code Snippets

Below are example code snippets and component structures for key elements, using **Next.js 15** (React 18) and **Tailwind CSS**. These are simplified for clarity, but illustrate how to structure the layout and apply styling/animation. The code is written as functional React components (assuming usage of Next.js App Router or Pages as appropriate) with Tailwind utility classes. You can adapt class names and structure as needed.

1. Navigation Bar (Header): A responsive header with logo and menu items. Uses Tailwind for layout; Heroicons for a mobile menu icon. In Next.js, you'd typically place this in a layout file or `_app.js`. This example shows a basic structure:

```
// components/Navbar.jsx
import { useState } from 'react';
import Link from 'next/link';
import { Bars3Icon, XMarkIcon } from '@heroicons/react/24/
outline'; // Heroicons for menu toggle

export default function Navbar() {
  const [menuOpen, setMenuOpen] = useState(false);
  const toggleMenu = () => setMenuOpen(!menuOpen);

  return (
    <nav className="w-full bg-white shadow-md fixed top-0 z-50">
      <div className="container mx-auto flex items-center justify-between p-4">
        {/* Logo / Brand Name */}
        <Link href="/" className="text-2xl font-serif text-primary">
          Deep Engineering
        </Link>
        {/* Desktop Menu */}
        <ul className="hidden md:flex space-x-8 text-gray-800 font-medium">
          <li><Link href="/about" className="hover:text-primary">About</Link></
li>
          <li><Link href="/technology" className="hover:text-
primary">Technology</Link></li>
          <li><Link href="/projects" className="hover:text-primary">Projects</
Link></li>
          <li><Link href="/team" className="hover:text-primary">Team</Link></li>
          <li><Link href="/contact" className="hover:text-primary">Contact</
Link></li>
        </ul>
        {/* Mobile Menu Button */}
        <button className="md:hidden p-2 text-gray-800" onClick={toggleMenu}
aria-label="Toggle Menu">
          {menuOpen ? <XMarkIcon className="h-6 w-6"/> : <Bars3Icon
className="h-6 w-6"/>}
```

```
        </button>
      </div>
      {/* Mobile Menu Dropdown */}
      {menuOpen && (
        <div className="md:hidden bg-white border-t border-gray-200">
          <ul className="flex flex-col px-4 py-2 space-y-2 text-gray-800">
            <li><Link href="/about" onClick={toggleMenu}>About</Link></li>
            <li><Link href="/technology" onClick={toggleMenu}>Technology</
Link></li>
            <li><Link href="/projects" onClick={toggleMenu}>Projects</Link></li>
            <li><Link href="/team" onClick={toggleMenu}>Team</Link></li>
            <li><Link href="/contact" onClick={toggleMenu}>Contact</Link></li>
          </ul>
        </div>
      )}
    </nav>
  );
}
```

Explanation: This Navbar uses a mobile-first approach. On medium screens (`md:` breakpoint) and up, it shows a horizontal menu. On small screens, it shows a hamburger icon (Bars3Icon) to toggle the menu. The menu links use Next's `<Link>` for client-side navigation. We apply a shadow and fixed positioning to mimic a sticky header. The `font-serif text-primary` on the logo suggests using the brand serif font and primary color (configured in Tailwind theme). The hover states change text to the primary color. This component is compatible with Next.js and Tailwind out-of-the-box. It will be easy to include in the main layout (e.g., in `app/layout.jsx` or `_app.js`).

2. Hero Section: A full-screen hero with text and call-to-action, utilizing Tailwind for styling and optionally Framer Motion for animation. Here's a component example:

```
// components/HeroSection.jsx
import { motion } from 'framer-motion';  // if using Framer Motion for animation

export default function HeroSection() {
  return (
    <section
      className="relative flex items-center justify-center text-center bg-cover
bg-center"
      style={{ minHeight: '100vh', backgroundImage: 'url("/hero-bg.jpg")' }}
    >
      <div className="bg-black bg-opacity-50 absolute inset-0"></div> {/*
overlay for contrast */}
      <div className="relative z-10 px-4">
        <motion.h1
          className="text-4xl md:text-6xl font-serif font-bold text-white"
          initial={{ y: 50, opacity: 0 }}
```

```
          animate={{ y: 0, opacity: 1 }}
          transition={{ duration: 0.8, ease: 'easeOut' }}
        >
          Continuous Clean Energy, Anywhere
        </motion.h1>
        <motion.p
          className="mt-4 text-lg md:text-2xl text-gray-100 max-w-2xl mx-auto"
          initial={{ y: 50, opacity: 0 }}
          animate={{ y: 0, opacity: 1 }}
          transition={{ delay: 0.3, duration: 0.8, ease: 'easeOut' }}
        >
          Delivering 24/7 renewable power with our revolutionary KPP technology
- no fuel, no emissions.
        </motion.p>
        <motion.div
          initial={{ opacity: 0 }}
          animate={{ opacity: 1 }}
          transition={{ delay: 0.8, duration: 0.5 }}
        >
          <a href="#technology"

className="inline-block mt-8 px-6 py-3 bg-primary text-white text-lg font-medium
rounded hover:bg-primary-dark"
          >
            Learn More
          </a>
        </motion.div>
      </div>
    </section>
  );
}
```

Explanation: This hero uses a full viewport height (`minHeight: 100vh`) with a background image (e.g., `hero-bg.jpg` in public folder – which could be an abstract energy-related image or video poster). We overlay a semi-transparent black layer to ensure white text is readable. The content (h1, p, button) is centered. We used Framer Motion's `<motion.h1>` etc., to animate the elements on mount (sliding upward from y:50 and fading in). The button is a simple anchor that could scroll to the Technology section (using a fragment link). The Tailwind classes give us responsive font sizing (text-6xl on md for larger screens) and spacing. We also show how to use a custom color class `bg-primary` and `hover:bg-primary-dark` – these would be defined in Tailwind config to correspond to our brand colors (e.g., primary = deep blue, primary-dark = slightly darker blue). The hero is set to **position relative** with children absolutely positioned where needed; we ensure the text is above the overlay by using z-index. This hero is easily integrated into a Next page (like pages/index.js or a homepage component).

3. Grid Layout (Technology Features Section): Using Tailwind's grid to layout feature cards or technical specs. For example, a section highlighting KPP features (Clean, Continuous, etc.) in a 3-column grid on desktop, 1-column on mobile:

```jsx
// components/FeaturesGrid.jsx
import { BoltIcon, GlobeAltIcon, ArrowsExpandIcon, DeviceMobileIcon } from
'@heroicons/react/24/outline';
// Example icons: Bolt for energy, Globe for anywhere, ArrowsExpand for
scalability, DeviceMobile as placeholder

export default function FeaturesGrid() {
  const features = [
    { icon: BoltIcon, title: 'Clean Power', desc: 'No fuel or emissions – 100%
green energy.' },
    { icon: GlobeAltIcon, title: 'Continuous 24/7', desc: 'Uninterrupted
baseload power, day and night.' },
    { icon: DeviceMobileIcon, title: 'Decentralized', desc: 'Installable at the
point of need, even off-grid.' },
    { icon: ArrowsExpandIcon, title: 'Scalable & Modular', desc: 'From a few MW
to 100+ MW, expandable on demand.' },
  ];
  return (
    <section className="py-16 bg-gray-50">
      <div className="container mx-auto px-4">
        <h2 className="text-3xl font-serif font-semibold text-center text-
gray-800 mb-12">Why KPP Technology?</h2>
        <div className="grid grid-cols-1 md:grid-cols-2 xl:grid-cols-4 gap-8">
          {features.map(({ icon: Icon, title, desc }) => (
            <div key={title} className="flex flex-col items-center text-center
p-6 bg-white rounded shadow-sm hover:shadow-md transition-shadow">
              <Icon className="h-12 w-12 text-primary mb-4" />
              <h3 className="text-xl font-semibold text-gray-800 mb-2">{title}</
h3>
              <p className="text-gray-600">{desc}</p>
            </div>
          ))}
        </div>
      </div>
    </section>
  );
}
```

Explanation: We defined an array of features with an icon, title, and description. The Tailwind grid classes create a responsive layout – 1 column on small, 2 on medium, 4 on x-large screens. Each feature card is centered, with some padding, white background, and a subtle shadow. On hover (desktop), we elevate the shadow to indicate interactivity (though these might not be clickable, it just adds a nice effect). Icons from Heroicons are used (outline style) with a fixed size and the brand primary color (`text-primary`). The section itself has a light gray background to distinguish it from other sections. We included a section heading “Why KPP Technology?” centered above the grid. This kind of component could be used on the

Home or Technology page to summarize benefits. It's easily customizable by changing icons or text, and the structure is clear.

4. Project Cards (Projects Page): Illustrating a grid of project cards using Tailwind. This shows how to loop through project data and create a consistent card layout:

```
// components/ProjectsList.jsx
const projects = [
  { name: 'Zakho 100MW', location: 'Kurdistan, Iraq', status: 'Planned – 2025',
description: 'Four 25MW KPP units to provide regional power.' },
  { name: 'Soran 100MW', location: 'Kurdistan, Iraq', status: 'Planned – 2025',
description: 'Baseload clean power for Northern Iraq.' },
  { name: 'Raparin 50MW', location: 'Kurdistan, Iraq', status: 'Planned –
2026', description: 'Expanding reliable energy access in Raparin.' },
  { name: 'Garmian 50MW', location: 'Kurdistan, Iraq', status: 'Planned –
2026', description: 'Sustainable power plant in development.' },
  { name: 'Samawah 90MW', location: 'Al-Muthana, Iraq', status: 'In
Development', description: 'Flagship KPP project with Board of Investment.' }
];

export default function ProjectsList() {
  return (
    <section className="py-16 bg-white">
      <div className="container mx-auto px-4">
        <h2 className="text-3xl font-serif font-semibold text-center text-
gray-800 mb-10">Our Projects</h2>
        <div className="grid grid-cols-1 md:grid-cols-2 xl:grid-cols-3 gap-8">
          {projects.map(project => (
            <div key={project.name}
className="border border-gray-200 rounded-lg p-6 hover:shadow-lg transition">
              <h3 className="text-2xl font-semibold text-primary
mb-1">{project.name}</h3>
              <p className="text-sm text-gray-600 mb-2">{project.location} –
<em>{project.status}</em></p>
              <p className="text-gray-700">{project.description}</p>
            </div>
          ))}
        </div>
      </div>
    </section>
  );
}
```

Explanation: This component creates a grid of project cards. We simulate project data in an array – in a real site this might come from a CMS. The grid goes from 1 column on mobile to 2 on md to 3 on xl, to nicely fill space. Each card has a light border and rounded corners. On hover, we add a larger shadow for a lift effect.

The project name is shown as a heading in the brand color, with location and status in a smaller, muted text. This demonstrates using Tailwind utility classes for spacing (`mb-1`, `mb-2` etc.) and text styling. The code is straightforward and highlights how content can be dynamically rendered from data. This layout is static-site friendly (just mapping data) and responsive. If needed, you could link each card to a detailed project page by wrapping `<Link href={`/projects/${slug}`}>` around the card contents.

5. Contact Form: Lastly, a contact form structure with Tailwind. This can be made to submit to an API route or third-party. Here's a simplified version (without actual submission logic):

```
// components/ContactForm.jsx
import { useState } from 'react';

export default function ContactForm() {
  const [formData, setFormData] = useState({ name: '', email: '', message:
'' });
  const [sent, setSent] = useState(false);

  const handleChange = e => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async e => {
    e.preventDefault();
    // TODO: send formData to an API or service
    setSent(true);
  };

  return (
    <section className="py-16 bg-gray-50" id="contact">
      <div className="container mx-auto px-4 max-w-lg">
        <h2 className="text-3xl font-serif font-semibold text-center text-
gray-800 mb-8">Get in Touch</h2>
        {!sent ? (
          <form onSubmit={handleSubmit} className="space-y-4">
            <div>
              <label className="block text-gray-700 mb-1" htmlFor="name">Name</
label>
              <input
                type="text" name="name" id="name" required
                className="w-full border border-gray-300 rounded px-3 py-2
focus:outline-none focus:border-primary"
                placeholder="Your Name"
                value={formData.name} onChange={handleChange}
              />
            </div>
            <div>
```

```
              <label className="block text-gray-700 mb-1"
htmlFor="email">Email</label>
              <input
                type="email" name="email" id="email" required
                className="w-full border border-gray-300 rounded px-3 py-2
focus:outline-none focus:border-primary"
                placeholder="you@example.com"
                value={formData.email} onChange={handleChange}
              />
            </div>
            <div>
              <label className="block text-gray-700 mb-1"
htmlFor="message">Message</label>
              <textarea
                name="message" id="message" rows="4" required
                className="w-full border border-gray-300 rounded px-3 py-2
focus:outline-none focus:border-primary"
                placeholder="How can we help you?"
                value={formData.message} onChange={handleChange}
              />
            </div>
            <button
              type="submit"
              className="w-full bg-primary text-white font-medium py-3 px-6
rounded hover:bg-primary-dark transition-colors"
            >
              Send Message
            </button>
          </form>
        ) : (
          <p className="text-center text-green-600 text-lg">Thank you! Your
message has been sent.</p>
        )}
      </div>
    </section>
  );
}
```

Explanation: This form collects name, email, and message. Tailwind is used to style labels and inputs. We give inputs a border and change border color on focus to the brand primary (for a clear focus state). The button is styled with the primary color and a hover state. The component manages local state just to show a success message on submission (in a real app, you'd integrate with an API or use a service like Formspree or a Next.js API route to handle emails). The form section is centered with a max width to not stretch too wide. All text is legible and the layout is mobile-friendly (full width fields). By using this as a controlled component with React state, we can easily integrate validation or sending logic. If connected to a CMS, the contact info (like recipient email) might be configured there, but the form itself is largely front-end code. The use of `id="contact"` also allows the hero "Learn More" button or nav links to scroll here if needed.

These code snippets illustrate a consistent approach: using Next.js Link for routing, Tailwind for utility-first styling (with a design system of colors/fonts configured), and integrating animation or interaction with React hooks and libraries. All components should be placed appropriately in the Next.js structure (e.g., pages or app directory). For deployment on **Vercel**, no special changes are needed – Next.js is Vercel's first-class citizen. For **cPanel hosting**, which typically doesn't run Node servers, we should build the site as a **static export**. That means using Next.js **Static Site Generation (SSG)** for all pages and avoiding runtime Node dependencies. The code above is compatible with SSG (no unauthorized server-side code). We'd run `next build && next export` to get static HTML/CSS/JS, which can then be uploaded to a cPanel host. Ensure that any dynamic data (e.g., from CMS) is fetched at build time (or use ISR – Incremental Static Regeneration – if on a platform that supports Node functions; but cPanel likely means pure static). Also, avoid features like Next.js API routes or Server Actions that require a Node environment, or replace them with external services (for the contact form, for instance, use a third-party or a serverless function on Vercel called via JS). By keeping the site mostly static, you ensure it runs on cPanel without issues. Tailwind CSS will purge and minimize the CSS on build, resulting in fast load times.

Pro Tip: When integrating these components, test each in isolation and as part of the whole site. Next.js 15 (if using App Router) might allow you to co-locate some components and possibly use new features (like React Server Components), but for simplicity, treat these as client components (especially ones with state or animation). Use the Next.js Image component for any actual `<img>` for optimized loading (just ensure to configure domains or use static imports for local images). Also configure your Tailwind theme (in `tailwind.config.js`) to include the brand colors (primary, etc.) and fonts (Crimson Pro, Heebo via @import or link). This way classes like `text-primary` or `font-serif` map to your desired styles.

By following these code patterns, developers can rapidly build out the site's front-end. The structure is modular (each section in its own component), and the styling is consistent via Tailwind. The result will be a site that is not only visually aligned with the design guide but also built with best practices for Next.js and modern CSS.

CMS Compatibility

To manage content efficiently, integrating a **Headless CMS** is recommended. The site should support visual editing and static site generation. Here are three CMS options with their benefits, any of which can work well with Next.js:

- **Sanity:** A flexible headless CMS with strong support for structured content and a customizable editing interface. Sanity would allow you to define schemas for each content type (pages, projects, team members, etc.). It pairs well with Next.js's static generation: you can fetch data at build time via Sanity's GROQ queries, and revalidate content on updates. Sanity recently introduced a **Visual Editing** mode that lets editors see their changes on the live site in real-time [31]. This means content creators can click on the site and edit content inline, bridging the studio and the site preview. Sanity's strengths include real-time collaboration (multiple editors), a rich text editor for long content, and the ability to handle references (e.g., a project entry can reference team members, etc.). You can host Sanity's backend for free up to certain usage, and it integrates with Next.js via official libraries (`@sanity/client`). For static export, you'd likely use Sanity's webhook to trigger a new build when content changes, since static files need rebuilding to update. Many Next.js projects use Sanity for its mix of developer flexibility and editor experience.

- **Storyblok:** A headless CMS known for its **visual editor** that is extremely user-friendly for non-developers. In Storyblok, content is organized in a component-based manner (bloks) which maps well to Next.js components. The Visual Editor loads your site in an iframe and allows editors to click on components to edit content, seeing changes live [32]. This is great for a marketing-heavy site or when the content structure might change often. For example, the marketing team could rearrange the homepage sections by reordering bloks in Storyblok. Storyblok also supports static site generation – you would use their API to fetch content at build time. They have a JS SDK and even example starters for Next. One advantage is that editors can create new pages or sections without developer intervention, if you set up the component model flexibly. It's a commercial SaaS (with a free tier for small sites), and it's very popular for its editor experience. If a **true WYSIWYG editing experience** is a priority, Storyblok is a top choice [33]. For deployment, similar to Sanity, you'd rebuild or use incremental static regeneration on content changes (Storyblok can ping a webhook on publish).

- **Netlify CMS (Decap CMS):** An open-source, git-based CMS that lives in your repo. This would add an `/admin` interface to the site where editors can log in (usually via GitHub or other OAuth) and edit Markdown/YAML content which lives in the repository. The benefit is simplicity and zero cost: content is stored as flat files, and when editors make changes, it commits to Git. This works with static site generation nicely because a new commit triggers a rebuild (if using Netlify or any CI/CD). Netlify CMS provides a fairly traditional form-based interface (not a visual inline editor), but it is user-friendly and can be configured for different collections (blog posts, projects, etc.). For a site like Deep Engineering, you could set up collections for Projects, Team, and static pages content (like About page text, which editors can then update). The editorial workflow (drafts, previews) is supported but not as instantaneous as Sanity/Storyblok. The advantage is that everything is version-controlled and there's no external DB – the site content is self-contained. This CMS might be suitable if you want to keep things simple and within Git, and if real-time visual editing is less important than having a straightforward editing form. Since it's called Decap CMS now (recent rebrand), it's still the same under the hood: a React app that publishes to git. With Next.js, you'd likely write a script to parse the markdown (or use `next-mdx-remote` or similar) at build time. There are plenty of examples of using Netlify CMS with Next.

- *(Honorable mentions):* **TinaCMS** is another visual editing toolkit that can provide inline editing on a Next.js site (Tina can be git-backed like Netlify CMS but with inline editing). Also **Contentful** or **DatoCMS** are headless CMS with good Next integration, though they don't have true visual editing (just preview). Since the prompt specifically mentioned Sanity, Storyblok, Netlify CMS, the above focus on those.

CMS Integration Considerations:

Whichever CMS you choose, structure your content models to match the site's IA. For example, in Sanity/Storyblok you might have a "Homepage" singleton with fields for the hero tagline, subtagline, etc., plus a list of references to feature blocks or project highlights. An "About" singleton with the company info and maybe an array for team members (or separate Team collection). For "Technology" pages, since there are multiple subpages, a flexible approach is needed – either each subpage is its own entry in a "TechPage" collection (with a type or slug identifying which aspect it is), or a single "KPP Technology" entry that holds all sub-sections (maybe not ideal for large content). Given the site's structure, treating each page (Overview, How it

Works, etc.) as separate entries might be clean. Then your Next.js pages can fetch those by a known slug or ID.

Static Generation Workflow: Ensure the CMS supports triggering rebuilds or using ISR: - With **Vercel**, you can use Next.js Incremental Static Regeneration. For example, set a `revalidate` on pages so they check for new content every X seconds, or better, use **On-Demand Revalidation**: configure the CMS to call a special revalidation API route on content publish (Next 12+ supports `res.revalidate('/page')` in API routes). That way, editors publish and the site updates without full redeploy. This works for Vercel hosting. - If you intend to **export to static** and host on GoDaddy cPanel, you won't have on-demand revalidation – you'd need to manually rebuild and upload, or have a CI pipeline that does this on CMS changes. In that scenario, a git-based CMS (Netlify CMS) might actually streamline things (because committing content triggers your build pipeline automatically via GitHub Actions or similar). Sanity/Storyblok would require a build hook set up on an external service (like GitHub Action that runs `next export` and deploys via FTP to cPanel, for example).

Visual Editing: Both Sanity and Storyblok offer visual editing where the site is loaded for preview and clicking text highlights the field in the CMS. For Sanity, you'd use their Content Source Map and perhaps their `@sanity/visual-editing` toolkit which connects the Next site to Sanity Studio [34]. Storyblok's visual editor is more out-of-the-box (just add the Storyblok script and it overlays edit buttons on your site). If having this feature is a priority for the client (so marketing folks can see exactly what they're doing), Storyblok might edge out others. However, Sanity's new Studio (v3) with Live Preview is also quite powerful, just a tad more setup.

CMS UI and Content Team: Consider who will use it. If the Deep Engineering team has technical folks, Sanity's structured content is fine. If they prefer a more WYSIWYG/page-builder feel, Storyblok could be better. Netlify CMS is okay for a smaller team comfortable with forms and can be self-hosted easily (just protect the admin with Netlify Identity or GitHub login).

Conclusion: Any of these CMS can be made to work with a Next.js static site; the decision will depend on budget, desired editing experience, and workflow. Storyblok and Sanity are proven with Next.js for static sites (many case studies exist). Sanity provides a robust backend with querying, which is great if you need to do more complex data relationships (it's almost like a database + CMS). Storyblok provides an excellent authoring experience with its visual composer [33], which might result in faster content iteration. Netlify CMS keeps everything in-house and simple, which is appealing for a straightforward content structure and if you want to avoid external services.

From a developer standpoint, initial setup for Sanity or Storyblok will take a bit of time (defining schemas or components), but once in place, they greatly ease content management and scaling the site (e.g., adding a new project is just filling a form). The site as planned (with projects, team, multi-page tech content) would benefit from a CMS to avoid hardcoding these into the Next.js code.

Recommendation: Evaluate the team's needs – if they want to frequently update technical content and see it as they edit, **Storyblok** might be the top choice for its intuitive visual editor [35]. If they value flexibility and might build more custom logic around content (or even use the content in other apps), **Sanity** is excellent and now also has visual editing capabilities. If low cost and simplicity are paramount, **Netlify (Decap) CMS** will do the job with a bit less polish in the editor UI. All options support static generation and will work with the design we outlined.

Regardless of choice, ensure the build and deployment pipeline is configured: for example, on Vercel, set up webhooks so that publishing in the CMS triggers `POST /api/revalidate` for the pages or triggers a new build if using static export; on a traditional hosting, you might rely on manual deploy unless you set up continuous deployment via Git.

By following this guide, the Deep Engineering website will be built to modern standards – aesthetically inspired by an industry leader (Vestas), with well-organized content and navigation, engaging yet professional copy, cohesive visuals, smooth interactive touches, clean code, and a content management setup that keeps the site maintainable long-term. The result will be a site that impresses visitors and effectively communicates Deep Engineering's technical prowess and vision for renewable energy.

---

1 2 3 4 5 7 8 9 10 11 13 17 21 22 23 24 25 26 27 28 Kinetic Power Plant (KPP) Technology – Technical Overview.html
file://file-TEMp4W3Ho8qkbckp4Rsgmh

6 KINETIC POWER PLANTS – HTL Solutions GmbH
https://www.htl-solutions.de/en/technologies/kinetic-power-plants/

12 Who We Are - Deep Engineering
http://www.deepengineering.co/who-we-are/

14 15 16 18 19 20 We offer KPP - Kinetic Power Plants
https://www.ki-tech.global/what-we-offer

29 Framer Motion vs GSAP for React Developers | by Aleksei Aleinikov
https://javascript.plainenglish.io/framer-motion-vs-gsap-for-react-developers-b6f71d1d5078

30 I Rebuilt 3 Awwwards Page Transitions using Nextjs and Framer ...
https://www.youtube.com/watch?v=WmvpJ4KX30s&pp=0gcJCfwAo7VqN5tD

31 Visual Editing with Sanity | Sanity Docs
https://www.sanity.io/docs/visual-editing/introduction-to-visual-editing

32 Visual Editor | Storyblok
https://www.storyblok.com/docs/editor-guides/visual-editor

33 Storyblok for Web Designers: Why We Use It for Next-Level Flexibility
https://www.afteractive.com/blog/storyblok-for-web-designers

34 Fetching content for Visual Editing | Sanity Docs
https://www.sanity.io/docs/visual-editing/fetching-content-for-visual-editing

35 What is a static site CMS? (5 benefits) - Storyblok
https://www.storyblok.com/mp/static-site-cms