Let's build **both** examples properly: one pure SSR page **(no Zustand)** and one **with Zustand** for user interactions.

**✅ First Concept Recap Before Code**

**1. Page 1 — CMS content page**

* Data is fetched from Contentful CMS on **each request** (SSR).
* **No Zustand** needed — page already gets fresh data at server side.
* If CMS content changes, **Next.js will automatically pick new content** on the **next page request** (because we use getServerSideProps).

OR  
If you want a **static site with rebuild after some time**, you can use **ISR (Incremental Static Regeneration)** with revalidate.

**2. Page 2 — Interactive form page**

* User enters data → e.g., a simple **Contact Us** form.
* We **use Zustand** here to manage form fields across components.

**🧩 Folder Structure**

|  |
| --- |
| /pages  /page1.tsx (SSR page - no Zustand)  /page2.tsx (Interactive page - uses Zustand)  /store  /formStore.ts (Zustand store for form)  /api  /cmsApiClient.ts (axios API call)  /types  /cmsTypes.ts (CMS data types)  /components  /ContactForm.tsx (form component using Zustand)  .env.local (should have NEXT\_PUBLIC\_API\_BASE\_URL) |

**📜 1. pages/page1.tsx (SSR page, CMS content, no Zustand)**

|  |
| --- |
| // pages/page1.tsx  import { GetServerSideProps } from 'next'  import { fetchCMSContent } from '@/api/cmsApiClient'  import { CMSContent } from '@/types/cmsTypes'  type Page1Props = {  content: CMSContent  }  export default function Page1({ content }: Page1Props) {  return (  <div>  <h1>{content.title}</h1>  <p>{content.body}</p>  </div>  )  }  // Server Side Rendering — fresh content on every page load  export const getServerSideProps: GetServerSideProps = async () => {  try {  const content = await fetchCMSContent()  return {  props: {  content,  },  }  } catch (error) {  console.error('Failed to fetch CMS content:', error)  return {  notFound: true,  }  }  } |

**📜 2. pages/page2.tsx (Interactive form page, uses Zustand)**

|  |
| --- |
| // pages/page2.tsx  import ContactForm from '@/components/ContactForm'  export default function Page2() {  return (  <div>  <h1>Contact Us</h1>  <ContactForm />  </div>  )  } |

**📜 3. components/ContactForm.tsx (Form with Zustand)**

|  |
| --- |
| // components/ContactForm.tsx  'use client'  import { useFormStore } from '@/store/formStore'  export default function ContactForm() {  const { name, email, message, setField, resetForm } = useFormStore()  const handleSubmit = (e: React.FormEvent) => {  e.preventDefault()  console.log('Form Submitted:', { name, email, message })  resetForm()  }  return (  <form onSubmit={handleSubmit}>  <input  type="text"  placeholder="Name"  value={name}  onChange={(e) => setField('name', e.target.value)}  />  <input  type="email"  placeholder="Email"  value={email}  onChange={(e) => setField('email', e.target.value)}  />  <textarea  placeholder="Message"  value={message}  onChange={(e) => setField('message', e.target.value)}  ></textarea>  <button type="submit">Submit</button>  </form>  )  } |

**📜 4. store/formStore.ts (Zustand form store)**

|  |
| --- |
| // store/formStore.ts  import { create } from 'zustand'  type FormState = {  name: string  email: string  message: string  setField: (field: keyof FormState, value: string) => void  resetForm: () => void  }  export const useFormStore = create<FormState>((set) => ({  name: '',  email: '',  message: '',  setField: (field, value) => set((state) => ({ ...state, [field]: value })),  resetForm: () => set({ name: '', email: '', message: '' }),  })) |

**📜 5. api/cmsApiClient.ts (CMS API client)**

|  |
| --- |
| // api/cmsApiClient.ts  import axios from 'axios'  import { CMSContent } from '@/types/cmsTypes'  const apiClient = axios.create({  baseURL: process.env.NEXT\_PUBLIC\_API\_BASE\_URL,  })  export async function fetchCMSContent(): Promise<CMSContent> {  const response = await apiClient.get('/cms/content')  return response.data  } |

**📜 6. types/cmsTypes.ts (Type for CMS data)**

|  |
| --- |
| // types/cmsTypes.ts  export type CMSContent = {  title: string  body: string  } |

**⚡ About revalidate (Incremental Static Regeneration - ISR)**

If you want **better performance** than SSR and **still keep CMS updates** fresh,  
you can use **getStaticProps with revalidate** like this:

|  |
| --- |
| export const getStaticProps: GetStaticProps = async () => {  try {  const content = await fetchCMSContent()  return {  props: {  content,  },  revalidate: 60, // Regenerate the page every 60 seconds if CMS content changed  }  } catch (error) {  console.error('Failed to fetch CMS content:', error)  return {  notFound: true,  }  }  } |

✅ **ISR (Incremental Static Regeneration)** means:

* The page is statically generated at **build time**.
* After 60 seconds (or whatever you set), **Next.js will regenerate the page** *in the background* if a request comes in.

✅ **User experience**:

* Always instant load.
* CMS updates will **automatically** reflect after the next revalidate time without needing full rebuild/deploy.

**🚀 Final Recommendations**

| **Situation** | **Suggestion** |
| --- | --- |
| CMS pages that rarely change | Use getStaticProps + revalidate. (ISR) |
| CMS pages that frequently change or are private | Use getServerSideProps (SSR). |
| User-interactive forms or session-based pages | Use Zustand or useState. |
| Pages needing very frequent real-time data (like stock prices) | Use client-side SWR, React Query. |

**🎯 In short:**

* Page 1 → **Server side data, no client-side state.**
* Page 2 → **Client-side interaction → use Zustand for clean local/global state.**
* Use **revalidate** for CMS updates if you want static generation but refresh after some seconds!