



NEXT.JS PROMPTBOOK

Ready-to-Use AI Prompts
for Everyday Coding Tasks

TABLE OF CONTENTS

Introduction

How to Use These Prompts?

Prompt for Styling & Components

Prompts for Schema & Data Layer

Prompts for Next.js Migration & Updates

Prompt for APIs & Backend

Prompt for Dynamic Pages & Routing

Prompts for Productivity & Debugging

Conclusion

INTRODUCTION

“As a CTO, I spend a lot of time looking for ways to speed up my work without sacrificing quality, and AI has been the answer to that.

Many developers worry about crafting perfect prompts, convinced that careful wording will guarantee the best results. The good news is, simple prompts in the right context are often more effective.

Below, you’ll see examples that stay short and clear, yet deliver strong outcomes because they build on real code or focus on small, well-documented features.

Adapt them to your own work, spend less time on repetitive fixes, and keep your projects moving.”

Happy prompting!

Jakub Dakowicz
CTO at Pagepro



FIND ME ON LINKEDIN

HOW TO USE THESE PROMPTS?

PROMPT ENGINEERING BEST PRACTICES

Designing great prompts takes more than copy-pasting. Here are some rules that'll get you the best results:

Be Explicit

The more context you give, like the framework version, coding style, project structure, the better the AI's response will be.

Provide Examples

Show the AI a snippet of what you already have so it can align with your style.

Break Big Tasks into Smaller Ones

Instead of asking for a whole feature at once, ask for a component, then a schema, then tests.

Iterate

Don't expect the first answer to be perfect. Treat it like pair-programming: ask, review, refine.

Check the Output

AI can make mistakes. Always validate the results before submitting them.

Experiment

Change variable names, add details about your stack, and re-ask the same questions with different wording.

WHICH AI TOOLS SHOULD YOU USE?

These are the AI solutions I use and the ways I integrate them into my work:

Cursor IDE

This has become my go-to coding environment. The built-in chat allows me to test ideas, debug issues, and get instant feedback without switching contexts. It's like having a coding partner on hand, ready to suggest improvements or catch mistakes before they slow me down.

Perplexity

When I need reliable research or want to dive deeper into a topic, I turn to Perplexity. It helps me gather references and get structured answers faster than traditional search. It's especially useful for staying ahead of new developments in frameworks and tools.

ChatGPT

For process design, refining copy, or sparking creative ideas, ChatGPT is where I go. I don't use it for writing production code, but it's invaluable for shaping documentation, brainstorming approaches, or simply seeing a problem from a different angle.

PROMPTS FOR STYLING & COMPONENTS

UPDATE STYLES: CONVERT FLEX TO GRID

LLMs are great in updating those small, but well-documented things that can take you forever.

This kind of prompt is handy if you need to test out different variants for your solution. For example converting flex to grid, change layouts to different (2 columns to 4, different alignments etc).



PROMPT IDEA

Convert these flex styles to the grid system.

We need 2 columns in a ratio of 3/1.

We need 2 rows, where 1st will have a 400px height and the bottom should have auto height.

Benefits:

Cleaner markup, better maintainability, and more control over responsive design.

Output Example:

Let me fix the final linting error:



IframeWrapper.tsx +1 •



No linter errors found in IframeWrapper.tsx

Perfect! I've successfully implemented the intersection observer loading for the iframe with a spinner. Here's what I've added:

Key Features Implemented:

- 1. Intersection Observer:** The iframe now loads only when it's close to the viewport (100px margin)
- 2. Loading States:** Three distinct states:
 - **Not loaded:** Shows a placeholder with spinner and "Loading form..." text
 - **Loading:** Shows an overlay spinner while the iframe loads
 - **Loaded:** Shows the iframe content
- 3. Performance Optimization:**
 - The iframe only starts loading when it's about to enter the viewport
 - Uses a 100px root margin to start loading slightly before it's visible
 - Proper cleanup of the intersection observer
- 4. User Experience:**
 - Uses the existing `Spinner` component from your project
 - Maintains the same visual styling and dimensions
 - Smooth transitions between loading states

UPDATE STYLES: CONVERT FLEX TO GRID

Instead of loading iframes right away, this prompt waits until they're close to the viewport.

While they're spinning up, it shows our existing loader so users aren't left staring at a blank box.



PROMPT IDEA

I need to load this iframe using the interesection observer when it will be close to the viewport.

Additionally I need to display a spinner when it's being loaded.

We already have a spinner in a project so use it instead of creating a new one.

Benefits:

Cleaner markup, better maintainability, and more control over responsive design.

ALTERNATIVE PROMPT SAMPLE



PROMPT IDEA

Convert this to use this

```
display: grid;  
grid-template-columns: 3fr 1fr;  
grid-template-rows: 3fr 1fr;
```

Benefits:

Cleaner markup, better maintainability, and more control over responsive design.

Output Example:

Summary of Changes

I've converted the component to use CSS Grid with the specified layout:

- Grid Container:** Changed from flexbox layout to CSS Grid using `grid grid-cols-1 lg:grid-cols-[3fr 1fr] grid-rows-1 lg:grid-rows-[3fr 1fr]`
- Grid Areas:**
 - **Main Content Card:** Takes up the large area (3fr width × 3fr height) - top-left
 - **Event Card:** Top-right area (1fr width × 3fr height)
 - **Search Card:** Bottom-left area (3fr width × 1fr height)
 - **CTA Card:** Bottom-right area (1fr width × 1fr height)
- Responsive Design:** Maintained mobile-first approach where cards stack vertically on smaller screens and use the grid layout on `lg` breakpoint and above
- Height Management:** Added `h-full` classes to ensure cards fill their grid areas properly and added minimum heights to the grid container
- Content Positioning:** Adjusted content positioning within cards to work well with the new grid layout, using flexbox for internal alignment

The layout now uses the exact CSS Grid specification you requested: `grid-template-columns: 3fr 1fr` and `grid-template-rows: 3fr 1fr`, creating a proportional 2x2 grid layout.

PROMPTS FOR SCHEMA & DATA LAYER

BUILD COMPONENT FROM FIGMA (WITHOUT SCHEMA)

Perfect for those early design-to-code steps: turn a Figma selection into a working React component without worrying about schemas yet. You can focus on the UI before layering in data structures.

Note: this should be done using the Figma MCP to connect to the design



PROMPT IDEA

Start by creating just a React component for this module.

Do not create the Sanity schema yet.

For text nodes, use the RichText component.

You can use @HeroMain.tsx as an example.

The image should have a similar style to the one in the hero main component.

Benefits:

Cleaner markup, better maintainability, and more control over responsive design.

Output Example:

✓ Component Features

1. Structure & Layout

- **Header Section:** Two-column layout with title and description
- **Tab Navigation:** Horizontal tabs with active states and keyboard navigation
- **Tab Content:** Full-width content area with background images and overlay text

2. Interactive Functionality

- **Tab State Management:** Uses `useState` to track active tab
- **Keyboard Navigation:** Supports Enter and Space key navigation
- **Accessibility:** Proper ARIA attributes and focus management

3. Styling & Design System

- **Typography:** Uses your existing CSS classes (`text-h5` , `text-section-subheading` , etc.)
- **Colors:** Matches your color palette (`primary-blue` , `secondary-dark-gray` , `tertiary-grey`)
- **Responsive Design:** Mobile-first approach with breakpoints
- **Image Overlay:** Uses your existing `ImageOverlay` component pattern

4. Key Design Patterns from HeroMain

- **Section wrapper** with conditional padding for first module
- **Container** with `max-width` and consistent padding
- **RichText** components for all text content
- **Proper TypeScript interfaces** following your conventions
- **Module props spreading** for Sanity integration

5. Responsive Behavior

- **Mobile:** Stacked layout with full-width tabs
- **Tablet/Desktop:** Side-by-side header layout with horizontal tabs
- **Tab content:** Maintains aspect ratio across all screen sizes

The component is ready to use and follows all your established patterns. When you're ready to create the Sanity schema, this component structure will integrate seamlessly with your CMS setup.

Would you like me to:

1. **Test the component** by adding it to your module map?
2. **Create the Sanity schema** for this module?
3. **Add any additional functionality** or styling adjustments?

BUILD COMPONENT & SCHEMA FROM FIGMA

When you need both the visual component and the Sanity schema, this prompt accelerates the full flow. Your UI and data model will stay in sync from the start.



PROMPT IDEA

I have a selection in Figma.

*I want a component that will match the one in Figma.
Use other modules as a reference.*

*Prepare a UI module component, and add it to the
modules list. Add it to the Sanity modules list.*

Create a Sanity schema for this module.

I want to name it ContentWithLogos.

*Title and descriptions should be a single simpleRichText
field.*

*I want a list of logos (min 3) and an option to turn on
animation.*

Benefits:

Cleaner markup, better maintainability, and more control over responsive design.

Output Example:

Perfect! I've successfully created the **ContentWithLogos** module that matches your Figma design. Here's what I've implemented:

✓ What's Been Created

1. Sanity Schema (`src/sanity/schemaTypes/modules/content-with-logos.ts`)

- **Content field:** Single `simpleRichText` field for title and description
- **Logos field:** Array of images with minimum 3 logos validation
- **Animation option:** Boolean toggle to enable/disable logo animation
- **Preview functionality:** Shows content preview with logo count and animation status
- **Initial values:** Pre-populated with "Our Partners" example content

2. React Component (`src/ui/modules/ContentWithLogos.tsx`)

- **Responsive design:** Mobile-first approach with proper spacing
- **Content section:** Centered text content using existing `RichText` component
- **Logo integration:** Uses your existing `LogosList` component with animation support
- **Consistent styling:** Follows project patterns with proper padding and container widths

3. Integration Updates

- ✓ Added to Sanity schema types index
- ✓ Added to modules fragments list (in "content" group)
- ✓ Added to React modules index with proper mapping
- ✓ Alphabetically ordered in all lists

Design Features

The component replicates your Figma design with:

- **Centered layout** with proper max-width containers
- **Two-part structure:** Content section + Logos section
- **Flexible content:** Rich text field supports the "Our Partners" styling pattern
- **Logo animation:** Optional horizontal scrolling animation
- **Responsive spacing:** Proper gaps and padding for all screen sizes

Usage

The module is now available in your Sanity Studio under the "Content" group. You can:

1. Add rich text content (title + description)
2. Upload minimum 3 partner logos
3. Toggle logo animation on/off
4. Use standard module options (padding, etc.)

The component will render exactly like your Figma design, with the logos displayed in a responsive grid (static) or animated horizontal scroll (when enabled).

REFACTOR SCHEMAS: EXTRACT COMMON CODE

Schemas often get messy with repeated patterns. Use this prompt to spot duplication and centralize shared fields into a base schema.



PROMPT IDEA

I have @link.tsx and @link.tsx - they both have a lot of common code.

Can we create a linkBaseSchema that can be used for both of them?

Benefits:

Cleaner codebase, easier maintenance, and fewer bugs caused by inconsistent schema definitions.

Output Example:

This refactoring:

1. **Creates a shared base** (`linkBaseSchema.ts`) with the common fields and preview select logic
2. **Eliminates code duplication** - the 4 core link fields (type, internal, external, params) are now defined once
3. **Maintains all existing functionality** - both schemas work exactly as before
4. **Makes future changes easier** - any updates to the core link logic only need to be made in one place
5. **Keeps schema-specific features** - the main link keeps its label and icon fields, richText link stays minimal

The base schema exports two functions:

- `createLinkBaseFields()` - returns the common field definitions
- `createLinkBasePreviewSelect()` - returns the common preview select object

Both existing schemas now extend this base while maintaining their specific differences.

I can see that both link schemas share a lot of common code. Let me analyze the differences and create a reusable base schema.

Read link.tsx

Read link.tsx

Perfect! I can see the differences between the two schemas:

Main differences:

1. **Name:** `"link"` vs `"richTextLink"`
2. **Label field:** Main link has a `label` field, richText link doesn't
3. **Icon fields:** Main link has `withIcon` and `icon` fields, richText link doesn't
4. **Preview:** Main link shows label/title and icon, richText link just shows "Link"
5. **Initial value:** Main link sets `withIcon: false`

Let me create a base schema that both can extend:

```
import { defineField, ReferenceRule, UrlRule } from "sanity";

import { makeVisibleFieldValidator, requiredField } from
"@/sanity/lib/validation-utils";
export const linkBaseFields = [
  defineField({
    name: "type",
    type: "string",
    options: {
      layout: "radio",
      list: [
        { title: "internal", value: "internal" },
        { title: "external", value: "external" },
      ],
    },
  }),
  defineField({
    name: "internal",
    type: "reference",
    title: "Internal Page",
    description: "Select a page within this domain",
    to: [{ type: "page" }, { type: "programme" }],
    hidden: ({ parent }) => parent?.type !== "internal",
    validation: makeVisibleFieldValidator<ReferenceRule>
(requiredField),
  }),
  defineField({
    name: "external",
    title: "External URL",
    description: "Enter a URL that links outside of our domain",
    placeholder: "https://example.com",
    type: "url",
    validation: makeVisibleFieldValidator<UrlRule>((Rule) =>
      Rule.uri({
        scheme: ["http", "https", "mailto", "tel"],
        allowRelative: true,
      })),
    hidden: ({ parent }) => parent?.type !== "external",
  }),
  defineField({
    name: "params",
    title: "URL parameters",
    placeholder: "e.g. #jump-link or ?foo=bar",
    type: "string",
    hidden: ({ parent }) => parent?.type !== "internal",
  }),
];
export const linkBasePreviewSelect = {
  _type: "internal._type",
  type: "type",
  title: "internal.title",
  internal: "internal.metadata.slug.current",
  params: "params",
  external: "external",
};
go(f, seed, [])
```

PROMPTS FOR NEXT.JS MIGRATION & UPDATES

UPDATE LEGACY MIDDLEWARE TO NEXT.JS 15+

Middleware APIs evolve quickly. Modernize old logic with this prompt, so it will run smoothly in Next.js 15 without breaking authentication or redirects.

```
import { NextResponse } from "next/server";
import type { NextMiddleware } from "next/server";

const middleware: NextMiddleware = async (request) => {
  const response = NextResponse.next();

  const {
    cookies,
    nextUrl: { pathname, locale },
  } = request;

  const locationCookieValue = cookies.get("locale")?.value;

  if (locale !== locationCookieValue) {
    const redirectUrl = request.nextUrl.clone();

    redirectUrl.locale = locale;

    return NextResponse.redirect(redirectUrl);
  }
  if (pathname === "/onlyLoggedInUsersCanAccessThisPath") {
    if (!cookies.get("user-session-cookie")) {
      const redirectUrl = request.nextUrl.clone();

      redirectUrl.pathname = "/login";

      redirectUrl.searchParams.set("returnUrl",
request.nextUrl.pathname);
      return NextResponse.redirect(redirectUrl);
    }
  }

  if (pathname.startsWith("/redirect-from-old-url-pattern")) {
    const { searchParams } = request.nextUrl;
    const pageId = searchParams.get("pageID");

    if (!pageId) {
      return; // show 404 page
    }

    const newUrl = request.nextUrl.clone();

    // redirectService - contains logic of converting page id to
    slugs
    const redirectRoute = await
redirectService.getNewUrlFromPageId<string>(
  pageId
);

    if (redirectRoute) {
      newUrl.pathname = redirectRoute;

      return NextResponse.redirect(newUrl);
    }
  }
};

export default middleware;
```



PROMPT IDEA

Update this syntax to be valid in NextJS 15.

Benefits:

Long-term compatibility, reduces tech debt, and keeps your app updated with the latest features.

UPDATE LEGACY API HANDLER TO NEXT.JS 15+

Old API handlers can break with new Next.js versions. With this prompt you can refactor them to follow updated patterns.

```
import { format } from "date-fns";

import { ApiRouteHandler } from "~api/types";
import PdfService from "~api/services/pdf";

const handler: ApiRouteHandler = async (request, response) => {
  try {
    const pdfService = new PdfService();

    const { pdfUrl } = await pdfService.createReport({
      userId: request.body.userId
    });

    const getPdfResult = await pdfService.getFile(pdfUrl);

    const filename = `Report_${format(new Date(), "yyyy-MM-dd")}.pdf`;

    response.setHeader("Content-Type", "application/pdf");
    response.setHeader(
      "Content-Disposition",
      `attachment; filename="${filename}"`
    );

    response.send(getPdfResult);
  } catch (error) {
    // Your error handling code here
  }
};

export default handler;
```



PROMPT IDEA

Can you update this syntax to Next.JS 15?

Benefits:





Long-term compatibility, reduces tech debt, and keeps your app updated with the latest features.

Output Example:




I've provided you with both versions updated for Next.js 15. The main issue remaining is the import path for your `PdfService` class. Here's what you need to do:

Summary of Changes Made:

App Router Version (Recommended - `app/api/generate-report/route.ts`)

-  Updated to use `NextRequest` and `NextResponse`
-  Changed to named export `POST` function
-  Updated request body parsing with `await request.json()`
-  Improved error handling

Pages Router Version (Compatible - `pages/api/generate-report.ts`)

-  Updated types to `NextApiRequest` and `NextApiResponse`
-  Added method checking for POST requests
-  Improved error handling

MIGRATE PAGE ROUTER CODE TO APP ROUTER

If you're still using `getStaticProps` and `getStaticPaths`, shift them into the modern App Router style with one prompt.

```
// In the news article page [id].js
export async function getStaticPaths() {
  return {
    paths: [],
    fallback: 'blocking', // or 'true' for incremental builds
  };
}

export async function getStaticProps({ params }) {
  const articleData = await fetchArticleData(params.id);
  return {
    props: { articleData },
    revalidate: 3600, // Revalidate every hour
  };
}
```



PROMPT IDEA

Convert this snippet for the app router in NextJS

Benefits:

New Next.js capabilities, simplifies data fetching, and keeps your code future-proof.

PROMPT FOR APIS & BACKEND

GENERATE API ROUTE FOR CONTACT FORM (SMTP + ERROR HANDLING)

Why build the same contact form handler over and over?
Get a ready-to-use route with SMTP support, proper branding, and Sentry logging baked in.



PROMPT IDEA

I need an API route for contact form handling.

I will have a form on the web with email & message fields.

I need an email template to handle these variables and it has to look friendly and use our branding colors.

It has to be submitted to the email from env variables: `CONTACT_EMAIL`.

It should use SMTP.

It should throw correct user-friendly errors and log them in Sentry.

Benefits:

New Next.js capabilities, simplifies data fetching, and keeps your code future-proof.

Output Example:

✓ Complete Contact Form System

1. API Route (`/api/contact`)

- **POST endpoint** that handles form submissions
- **Input validation** using Zod schema (email format, message length 10-1000 chars)
- **SMTP email sending** with configurable providers
- **Comprehensive error handling** with user-friendly messages
- **Sentry integration** for error logging and monitoring

2. Email Template

- **Richfield branding** using your official colors:
 - Primary Blue (`#0056AF`) for headers and accents
 - Primary Red-Orange (`#DE473C`) for highlights
 - Professional typography with Lato and Oswald fonts
- **Responsive design** that works on all devices
- **Clean layout** with proper spacing and visual hierarchy
- **Reply-to functionality** so you can respond directly to the sender

3. Contact Form Component (`ContactForm.tsx`)

- **Real-time validation** with immediate feedback
- **Loading states** and success confirmation
- **Accessibility features** (proper labels, ARIA attributes)
- **Error handling** for network issues and validation errors
- **Responsive design** using your Tailwind CSS classes

4. Environment Configuration

- **Updated `env.ts`** with contact form variables
- **Required environment variables:**
 - `CONTACT_EMAIL` - where to send the messages
 - `SMTP_HOST`, `SMTP_PORT`, `SMTP_USER`, `SMTP_PASS` - SMTP configuration

5. Comprehensive Documentation

- **Complete setup guide** in `docs/CONTACT_FORM.md`
- **SMTP provider instructions** (Gmail, Outlook, Yahoo, custom)
- **Troubleshooting section** for common issues
- **Security considerations** and future enhancement ideas

PROMPT FOR DYNAMIC PAGES & ROUTING

GENERATE DYNAMIC PAGE FOR MEDIA CONTENT

Set up a page structure with slug handling, metadata, and placeholders, so you can expand it as needed.



PROMPT IDEA

*I need to create a new route for products/[slug]
It will use new document types - products.*

*Create a schema for product in Sanity. Keep it simple,
just title, metadata, slug so we can generate route and
fetch sample data.*

Slug will look like this: /products/:id-of-product

*Create a route file with empty page template that will
jsut display layout + title.*

*Prepare methods like generatemetadata and fetch
sample data from Sanity (products schema) on the
server side.*

*These pages will be SSG pages. We need to update
revalidate API route to handle this pages too.*

Benefits:

Speeds up new feature rollout, enforces consistency across routes, and reduces mistakes with boilerplate code.

PROMPTS FOR PRODUCTIVITY & DEBUGGING

DEBUG SCHEMA ERRORS WITH SANITY TYPEGEN

Sanity typegen errors can be vague and frustrating, but you don't need to look for a solution by hand!

AI can help solve any console errors, so give it a try with this prompt for finding the differences that can break your schemas.

```
> richfield@1.0.0 predev /Users/jakub/Projects/richfield-college-website
> pnpm ts:typegen

> richfield@1.0.0 ts:typegen /Users/jakub/Projects/richfield-college-website
> pnpm ts:extract && pnpm ts:generate

> richfield@1.0.0 ts:extract /Users/jakub/Projects/richfield-college-website
> sanity schema extract

⚠ Extracting schemaYou have configured Sanity client to use a token in the browser. This may cause unintentional security
issues. See https://www.sanity.io/help/js-client-browser-token for more information and how to hide this warning.
✖ Failed to extract schema

SchemaError: SchemaError
    at ~/Projects/richfield-college-website/node_modules/.pnpm/sanity@4.1.1_@emotion+is-prop-valid@1.2.2_@types+node@24.1.0_@type
    at Open file in editor (cmd + click) ~/Projects/richfield-college-website/node_modules/sanity/lib/index.js:68484:17
    at ~/Projects/richfield-college-website/node_modules/.pnpm/sanity@4.1.1_@emotion+is-prop-valid@1.2.2_@types+node@24.1.0_@type
    at ~/Projects/richfield-college-website/node_modules/.pnpm/sanity@4.1.1_@emotion+is-prop-valid@1.2.2_@types+node@24.1.0_@type+react-dom@19.1.6_@types+r_ei4yvmqchfrr2xloak6fsj3mq4/node_modules/sanity/lib/index.js:68458:11
    at Array.map (<anonymous>)
    at prepareConfig (~Projects/richfield-college-website/node_modules/.pnpm/sanity@4.1.1_@emotion+is-prop-valid@1.2.2_@types+node@24.1.0_@types+react-dom@19.1.6_@types+r_ei4yvmqchfrr2xloak6fsj3mq4/node_modules/sanity/lib/index.js:68442:34)
    at Object.resolveConfig (~Projects/richfield-college-website/node_modules/.pnpm/sanity@4.1.1_@emotion+is-prop-valid@1.2.2_@types+node@24.1.0_@types+react-dom@19.1.6_@types+r_ei4yvmqchfrr2xloak6fsj3mq4/node_modules/sanity/lib/index.js:68951:7)
    at Object.getStudioWorkspaces (~Projects/richfield-college-website/node_modules/.pnpm/sanity@4.1.1_@emotion+is-prop-valid@1.2.2_@types+node@24.1.0_@types+react-dom@19.1.6_@types+r_ei4yvmqchfrr2xloak6fsj3mq4/node_modules/sanity/lib/_chunks-cjs/getStudioWorkspaces.js:43:92)
    at main (~Projects/richfield-college-website/node_modules/.pnpm/sanity@4.1.1_@emotion+is-prop-valid@1.2.2_@types+node@24.1.0_@types+react-dom@19.1.6_@types+r_ei4yvmqchfrr2xloak6fsj3mq4/node_modules/sanity/lib/_internal/cli/threads/extractSchema.js:10:50)
    at Object.<anonymous> (~Projects/richfield-college-website/node_modules/.pnpm/sanity@4.1.1_@emotion+is-prop-valid@1.2.2_@types+node@24.1.0_@types+react-dom@19.1.6_@types+r_ei4yvmqchfrr2xloak6fsj3mq4/node_modules/sanity/lib/_internal/cli/threads/extractSchema.js:25:1)
```



PROMPT IDEA

Can you check in diff what can make the schema incorrect?

Benefits:


Speeds up new feature rollout, enforces consistency across routes, and reduces mistakes with boilerplate code.



Output Example:

Schema Issues Identified:

1. Duplicate Module Entry (Primary Issue)

In `src/sanity/schemaTypes/fragments/modules.ts`, line 16 has a duplicate `"card-tabs"` entry:

 typescript

  Apply to index.ts

```
of: [  
  { type: "card-list" },  
  { type: "card-tabs" }, // Line 9  
  { type: "content-with-logos" },  
  // ... other modules  
  { type: "card-tabs" }, // Line 16 - DUPLICATE!  
],
```

GENERATE TAB SECTION STYLEGUIDE WITH SAMPLE DATA

In our projects, we include a `styleguide` page that shows the sample usage of some of the components. Styleguides often need multiple usage examples. This prompt generates tabs with varied content scenarios, saving you the manual setup.



PROMPT IDEA

Create a Tab section styleguide based on other styleguide sections.

Use @tab component and do not implement your own styles.

Create sample data, but omit the images. Use schema initialValues as a reference for content.

Create multiple scenarios: 1 tab, 3 tabs, 5 tabs.

Mix the possibilities of content inside (text only, text + image, image only).

Benefits:

Easier component testing, better documentation, and faster onboarding for new team members.

Output Example:

This updated tabs styleguide now:

Uses Real Components:

- `CardTabSection` - The complete section component from `@/ui/modules/CardTabs`
- `Tabs` - The standalone tabs component from `@/ui/modules/CardTabs/Tabs`

Shows Real Data Structures:

- Uses proper `Sanity.CardTabs` types
- Shows the actual rich text content structure
- Demonstrates the image overlay functionality

CONCLUSION

"The goal of my ebook was to give you practical tools you can apply directly in your daily work."

Remember: AI won't replace your skills or judgment.

It works best as a partner that helps you move past repetitive tasks and focus on solving the challenges that really matter.

At Pagepro, we've seen how combining strong engineering practices with AI can help companies modernize their platforms, optimize performance, and scale without unnecessary complexity.

If you'd like to explore how we can support your team, whether it's a CMS migration, a Next.js project, or building a React Native app, let's meet!"

Jakub Dakowicz
CTO at Pagepro



BOOK A MEETING

THANKS FOR READING!

EXPLORE OUR
NEXT.JS SERVICES



pagepro.co