Strapi vs Contentful

Here’s a crisp, decision-ready comparison of **Strapi vs Contentful**—with prices (what’s public) and the features that tend to matter in real builds.

**TL;DR**

* **Strapi** = open-source, self-hostable headless CMS with optional Strapi Cloud. Great when you want code-level control, SQL database choice, and to own your infra. Free OSS core; Cloud has Free + paid tiers; enterprise/self-host quotes. ([GitHub](https://github.com/strapi/strapi?utm_source=chatgpt.com))
* **Contentful** = fully-managed SaaS headless CMS with strong governance, app framework, Functions, and enterprise options (incl. EU data residency). Free plan exists; paid plans (Lite/Premium) are sales-assisted. ([Contentful](https://www.contentful.com/pricing/" \o "Contentful Pricing | Contentful))

**Pricing snapshot (public info + reputable trackers)**

⚠️ Note: both vendors adjust pricing and often keep enterprise numbers off-site. Treat third-party figures as **indicative**, and confirm with sales for a quote.

| **Platform** | **Free plan** | **Published paid tiers** | **Indicative prices seen on trackers\*** |
| --- | --- | --- | --- |
| **Strapi** | **Yes** (OSS Community; Strapi Cloud Free) | Cloud tiers (e.g., Essential/Pro/Scale) + self-hosted Enterprise (quote) | Trackers commonly show Cloud at ~US$29 (Dev) / US$99 (Pro) / US$499 (Team) per project/month; self-hosted Enterprise sometimes listed ~US$99/seat—**verify with Strapi**. ([GitHub](https://github.com/strapi/strapi?utm_source=chatgpt.com)) |
| **Contentful** | **Yes** (Platform “Free”) | **Lite** and **Premium** (quote) | Older public tiers started around ~US$300/mo on trackers; enterprise has add-ons (e.g., Gold Support+Uptime SLA bundle listed ~US$22k/yr after discounts)—**verify with Contentful**. ([Contentful](https://www.contentful.com/pricing/" \o "Contentful Pricing | Contentful)) |

\* Third-party trackers are useful for budgeting but may lag or differ from your negotiated quote.

**Useful inclusions called out publicly**

* Contentful Free shows **2 roles, 2 locales, 100k API calls/mo**, and **50 GB CDN** (no overages). ([Contentful](https://www.contentful.com/pricing/" \o "Contentful Pricing | Contentful))
* Strapi Cloud announced a **Free plan** in May 2025; docs show **Free + 3 paid plans**. ([Strapi](https://strapi.io/blog/introducing-the-free-plan-for-strapi-cloud?utm_source=chatgpt.com" \o "Introducing the Strapi Cloud Free Plan: Deploy Any Project for $0))

**Feature comparison**

| **Area** | **Strapi** | **Contentful** |
| --- | --- | --- |
| **Hosting model** | Self-host (MIT OSS) or Strapi Cloud (managed PaaS). Choose your infra or let them host. ([GitHub](https://github.com/strapi/strapi?utm_source=chatgpt.com)) | Fully managed SaaS; spaces live on Contentful’s platform. ([Contentful](https://www.contentful.com/pricing/" \o "Contentful Pricing | Contentful)) |
| **APIs** | REST by default; **GraphQL via plugin** (Strapi 5 updated GraphQL with Relay-style options). ([Strapi 5 Docs](https://docs.strapi.io/cms/plugins/graphql?utm_source=chatgpt.com" \o "GraphQL plugin | Strapi 5 Documentation)) | REST + **GraphQL Content API** out of the box. ([Contentful](https://www.contentful.com/developers/docs/references/graphql/?utm_source=chatgpt.com" \o "GraphQL Content API | Contentful)) |
| **Extensibility** | Plugin architecture; full code-level customization (Node/TypeScript). ([Strapi](https://strapi.io/features?utm_source=chatgpt.com" \o "Full-featured headless CMS and customizable API | Strapi)) | **App Framework** (UI extensions, apps), **Functions** (serverless) for backend logic; rich webhooks. ([Contentful](https://www.contentful.com/developers/docs/extensibility/app-framework/?utm_source=chatgpt.com" \o "App Framework - Contentful)) |
| **Editor UX** | Modern admin (Content-Type Builder, Draft/Publish, i18n plugin). Highly customizable. ([Strapi 5 Docs](https://docs.strapi.io/cms/intro?utm_source=chatgpt.com" \o "Welcome to the Strapi CMS Documentation!)) | Strong governance/UI; newer **Studio** and personalization products integrate with Platform. ([Contentful](https://www.contentful.com/pricing/" \o "Contentful Pricing | Contentful)) |
| **Governance & security** | You control stack/security posture when self-hosting; SOC 2 and GDPR noted; role-based permissions. DB lives where you deploy. ([Strapi](https://strapi.io/pricing" \o "Strapi - Pricing Cloud)) | Enterprise-grade options: SSO/SCIM, custom roles, SLAs, **EU data residency** add-on (Premium), multi-region delivery infra. ([Contentful](https://www.contentful.com/pricing/" \o "Contentful Pricing | Contentful)) |
| **Databases / data model** | SQL DBs (PostgreSQL, MySQL/MariaDB, SQLite). **No MongoDB**. Strapi Cloud ships with Postgres by default. ([Strapi 5 Docs](https://docs.strapi.io/snippets/supported-databases?utm_source=chatgpt.com" \o "supported-databases | Strapi 5 Documentation)) | Managed store—no DB to run. You consume via APIs; CDN/infra handled for you. ([Contentful](https://www.contentful.com/developers/docs/references/?utm_source=chatgpt.com" \o "API reference | Contentful)) |
| **Localization** | i18n plugin (self-host) and Cloud support, configurable. ([Strapi 5 Docs](https://docs.strapi.io/cms/intro?utm_source=chatgpt.com" \o "Welcome to the Strapi CMS Documentation!)) | Locales per plan; Free lists **2 locales**; higher plans expand. ([Contentful](https://www.contentful.com/pricing/" \o "Contentful Pricing | Contentful)) |
| **API limits** | Self-host: you set limits. Cloud: usage-based plan limits (see pricing page/docs). ([Strapi 5 Docs](https://docs.strapi.io/cloud/getting-started/usage-billing?utm_source=chatgpt.com" \o "Information on billing & usage | Strapi 5 Documentation)) | Clear quotas on Free (e.g., **100k API calls/mo**, **50 GB CDN**); more on paid. ([Contentful](https://www.contentful.com/pricing/" \o "Contentful Pricing | Contentful)) |

**Which should you pick?**

Choose **Strapi** if you want:

* **Full control** over code, database, and hosting (own your infra, keep data in your VPC). ([GitHub](https://github.com/strapi/strapi?utm_source=chatgpt.com))
* To run on **SQL** (Postgres/MySQL/SQLite) and customize business logic deeply. ([Strapi 5 Docs](https://docs.strapi.io/snippets/supported-databases?utm_source=chatgpt.com" \o "supported-databases | Strapi 5 Documentation))
* A **low cost of entry** (OSS) and you’re comfortable operating infra—or you’ll use **Strapi Cloud** for convenience. ([Strapi](https://strapi.io/blog/introducing-the-free-plan-for-strapi-cloud?utm_source=chatgpt.com" \o "Introducing the Strapi Cloud Free Plan: Deploy Any Project for $0))

Choose **Contentful** if you want:

* **Zero-ops SaaS** with strong governance, SLAs, and enterprise controls (SSO/SCIM, custom roles). ([Contentful](https://www.contentful.com/pricing/" \o "Contentful Pricing | Contentful))
* Extensibility with **App Framework**, **Functions**, and webhooks—without running your own servers. ([Contentful](https://www.contentful.com/developers/docs/extensibility/app-framework/?utm_source=chatgpt.com" \o "App Framework - Contentful))
* **EU data residency** (Premium add-on) or multi-region delivery features out of the box. ([Contentful](https://www.contentful.com/developers/docs/infrastructure/eu-data-residency/?utm_source=chatgpt.com" \o "EU data residency - Contentful))

**Practical cost guidance (how teams budget)**

* For **Contentful**, most non-trivial teams land on paid tiers; public site shows **Free / Lite / Premium** without dollar amounts. Trackers historically list **~US$300+/mo** starting points for older mid-tier plans, and enterprise add-ons priced separately; negotiate based on volumes (API calls, spaces, users, SLAs). **Always confirm with sales.** ([Contentful](https://www.contentful.com/pricing/" \o "Contentful Pricing | Contentful))
* For **Strapi**, you can run **free** on your own infra (CAPEX/OPEX = your hosting + ops). **Strapi Cloud** adds managed hosting; plans are **Free + paid tiers** (names vary). Trackers frequently show **US$29/99/499** as common reference points—but treat as ballpark. ([GitHub](https://github.com/strapi/strapi?utm_source=chatgpt.com))

**Recommendation template**

If you share your **team size**, **traffic/API volume**, **compliance needs** (e.g., EU residency), and whether you prefer **self-host vs SaaS**, I’ll map you to a plan tier and provide a ready-to-implement architecture (CDN, build pipeline, auth, and cost forecast).

Small team of 1 core dev, 3 contributors, no real compliance needs, building an educational investment website with low levels of change, a few key education pathways, freemium subscription model integrating aarc capital's proprietary option pricing and portfolio optimisation tools at the paid tiers. We can roll our own infrastructure, but thinking vercel and nextjs for website.

Awesome—thanks for the detail. For **1 core dev + 3 contributors**, low content churn, Next.js on Vercel, and a freemium model with proprietary quant tools, here’s a practical, low-ops plan.

**Recommendation (short)**

* **Pick: Strapi (self-hosted) + Next.js (Vercel)**  
  You’ll keep costs tiny, own your data model, and avoid SaaS API caps. Content changes are rare, so you can rely on ISR and webhooks.
* **Alternative if you want zero-ops right now:** Contentful Free → upgrade only if traffic or governance grows.

**Why Strapi fits your case**

* **Cost & control:** Free OSS core; no per-API-call limits. You can model “education pathways,” lessons, modules, and paywalled assets exactly how you want.
* **Low churn + ISR:** Pages regenerate on change (Incremental Static Regeneration) so you get the speed & SEO of static with minimal server costs.
* **Integration flexibility:** Easy to wire your **AARC pricing/optimisation tools** via your own microservice/API without worrying about CMS platform restrictions.
* **Auth & paywalls:** Keep subscription logic outside the CMS (Stripe + your user store), and use CMS only for content & visibility flags.

**If you prefer Contentful**

* **Pros:** true zero-ops; generous Free plan; nice editorial UX.
* **Trade-offs:** API quotas on free tiers, pricing jumps as you add users/locales/api calls, and less code-level control. For your “few pathways, low updates” scenario it’s fine—but the paywall/integration logic lives outside Contentful anyway, so you’re not gaining much vs Strapi.

**Proposed architecture**

**Services**

1. **Frontend:** Next.js on Vercel
   * Rendering: **ISR** for all public pages (pathways, lessons); **SSR** only for account pages.
   * Revalidation: webhook from CMS → /api/revalidate to refresh specific routes.
   * Feature gating: middleware checks subscription tier → route segment protection.
2. **CMS:** **Strapi** (self-hosted)
   * Host on **Fly.io / Render / Railway / your cloud** (small Postgres + a single Strapi app instance).
   * Content model (minimum viable):
     + Pathway (slug, title, summary, order, tier: free/premium)
     + Module (belongs to Pathway, order)
     + Lesson (belongs to Module; body (Rich Text/MDX), assets, estimated time)
     + GlossaryTerm (optional)
     + PromoBanner / Announcement (optional)
   * Roles: Admin (you), Editor (contributors).
   * Fields for SEO (title/description/og image) baked in.
3. **AARC tools service:** containerised microservice (Node or Python)
   * Endpoints: /price-options, /optimize-portfolio
   * Auth: service token + user JWT from frontend
   * Queue (if heavy): lightweight worker (e.g., BullMQ/Redis or a managed queue) for long-running jobs; frontend polls job status.
4. **Auth & billing:**
   * **Stripe** for subscriptions (free vs premium tiers).
   * **Auth** via NextAuth/Auth.js with email/pass or OAuth; store users in Postgres (Supabase/Neon/Vercel Postgres).
   * Webhooks: Stripe → upgrade/downgrade → mark user entitlement in your user DB; not in CMS.

**Data flows**

* **Publish content** in Strapi → webhook → Vercel revalidate (just the affected routes).
* **Premium lesson request** → Next.js middleware checks entitlement → if allowed, fetch lesson content + call AARC microservice as needed.
* **SEO**: ISR prebuilds key pages; sitemap generated nightly (cron on Vercel).

**Rough costs (ballpark, monthly)**

* **Vercel**: $0–$40 (Hobby/Pro) depending on usage.
* **Strapi**: $5–$30 app + $5–$15 Postgres on Fly/Render/Railway (dev + tiny prod).
* **AARC service**: $5–$30 depending on compute.
* **Stripe**: pay-as-you-go fees only.

Many teams run this stack comfortably **<$100/mo** at early traffic.

**Build checklist (2–3 dev days to MVP)**

1. **Spin up Strapi** (Postgres) and define: Pathway/Module/Lesson (+ tier flags).
2. **Seed content** for 1–2 pathways.
3. **Next.js pages**: /pathways/[slug], /lessons/[slug] (static, ISR).
4. **Revalidation route**: /api/revalidate + Strapi webhook.
5. **Auth + Stripe**: basic checkout → set user.tier.
6. **AARC service**: deploy minimal /price-options endpoint; secure with service token.
7. **Paywall**: middleware checks user.tier; hide premium modules in UI for free users.
8. **Analytics & SEO**: canonical tags, sitemap, simple event tracking.
9. **Editor training**: 30-min run-through of “create module/lesson → publish”.

**When to consider switching to Contentful later**

* Non-dev editors demand stricter governance/approvals, or you need multi-space workflows.
* You hit internal bandwidth limits and want to eliminate CMS hosting/patching entirely.
* You need vendor SLAs and advanced enterprise features.

If you want, I can sketch the **exact Strapi content model**, the **Next.js route structure**, and a **webhook revalidation snippet** you can paste in now.

**Next.js + Vercel**

**Rapid UI prototyping in Figma via Shadcn or heroui component design guides**

open source web application technology stack that uses next.js, vercel, headless ui, ( e. like ark-ui, or headless.ui), tailwind css, and leverages a component library such a shadcn or heroui to rapidly implement figma prototypes via figma mcp

I've created a comprehensive tech stack guide for building Next.js applications from Figma prototypes. Here are the key highlights:

**Key Features:**

1. **Semantic Naming Philosophy**: Components are named by purpose (ProductCard, HeroSection) rather than properties (BlueButton, FlexColumn)
2. **Design Token System**: Centralized tokens for colors, spacing, typography, and radii to maintain consistency
3. **Combo Classes**: Reusable CSS class combinations for common patterns (page-container, heading-primary, button-primary)
4. **Structured Directory**: Clear component hierarchy (ui → shared → features → sections → layout)
5. **Comprehensive Cursor Rules**: Detailed guidelines for AI-assisted development including:
   * Naming conventions
   * Component structure
   * Figma-to-code translation
   * Code review checklists
6. **Figma MCP Configuration**: Settings for automated design token extraction and semantic component naming

**Why This Approach Works:**

* **Maintainability**: Semantic names make code self-documenting
* **Scalability**: Design tokens and combo classes ensure consistency as the project grows
* **Speed**: shadcn/ui + combo classes enable rapid prototyping
* **Quality**: TypeScript + coding standards reduce bugs
* **Collaboration**: Clear conventions help teams work together

The setup is optimized for both human developers and AI assistants like Cursor, making it ideal for rapid Figma-to-production workflows.

Great news! There are existing Figma MCP servers that work with Cursor, but let me create a customized configuration and wrapper that integrates specifically with your tech stack requirements (Next.js, shadcn/ui, semantic naming, etc.).Perfect! I've created a comprehensive guide for setting up Figma MCP with Cursor, specifically configured for your tech stack. Here's what's included:

**What You Get:**

**1. Official Figma MCP Setup**

* Uses Figma's official MCP server that works with Cursor and other AI coding tools
* Step-by-step installation and configuration
* Authentication setup with personal access tokens

**2. Custom Cursor Rules**

Detailed rules that teach Cursor how to:

* Transform Figma frame names into semantic component names
* Extract design tokens automatically
* Map Figma auto-layouts to Tailwind classes
* Choose appropriate shadcn/ui components
* Generate responsive, accessible code

**3. Translation Mapping**

Automatic conversion of:

* Figma Auto Layout → Flexbox/Grid
* Figma Text Styles → Typography tokens
* Figma Color Styles → HSL design tokens
* Figma Components → shadcn/ui components

**4. Workflow Examples**

Ready-to-use prompts like:

@figma Fetch the "Product Card" frame

Generate a ProductCard component following our tech stack

**5. Optional Automation Script**

TypeScript helper that streamlines the Figma-to-component workflow

**How It Works:**

1. **You paste a Figma URL** or reference a component name
2. **Cursor fetches the design data** via MCP
3. **AI analyzes the structure** and applies your tech stack rules
4. **Generated code follows your standards**: semantic naming, design tokens, shadcn/ui components, TypeScript types

The key innovation is the **detailed Cursor rules** that ensure every generated component follows your conventions automatically—no more "Frame123" or "BlueButton" nonsense!

Would you like me to create additional templates for specific component types (cards, forms, navigation, etc.) or help you set this up with a specific Figma file?

Here’s a single, decision-ready **UI Tech-Stack document** for the **AARC Webtools** project that merges your requirements (Next.js + Vercel, Tailwind, shadcn **or** HeroUI, headless model), **Strapi** as the CMS, and **Figma MCP integration in Cursor** to auto-generate UI from prototypes. It also bundles a **.cursorrules** set for semantic naming + combo classes.

**AARC Webtools — UI Tech-Stack & Workflow**

**1) Goals**

* Prototype fast in **Figma**, then **auto-generate high-quality Next.js components** using Cursor’s MCP + our rules.
* Keep UI library **swappable** (shadcn or HeroUI) behind an abstraction; style with Tailwind and **combo classes** for consistency.
* Use **Strapi** as headless CMS; keep paywall / quant tooling separate from CMS.
* Deploy on **Vercel**; use ISR + targeted revalidation webhooks for low-churn content.

**2) Core Stack**

* **Framework:** Next.js 14+ (App Router) • **Hosting:** Vercel • **Styling:** Tailwind CSS • **UI Primitives:** Radix via **shadcn/ui** (primary) OR **HeroUI/NextUI** (alt) • **Design→Code:** **Figma MCP** in Cursor • **CMS:** **Strapi** (headless)

**Project layout (baseline):**

src/

app/ # Next.js routes (App Router)

components/

ui/ # Abstraction layer -> shadcn or HeroUI

layout/ # Header, Footer, Sidebar

sections/ # Hero, Features, Pricing, etc.

features/ # Domain features (OptionPricer, Optimizer, etc.)

shared/ # Spinner, ErrorBoundary, etc.

lib/

tokens.ts # Design tokens

design-system.config.ts

utils.ts

styles/

globals.css # Tailwind + combo classes (+ design systems)

types/

.cursorrules

figma.config.json

**3) Multi-Library UI Abstraction (shadcn ↔ HeroUI)**

We export **one import surface** (@/components/ui) and swap implementations. This lets you try **shadcn** templates first, but switch to **HeroUI** for motion-rich screens without refactors.

**Install HeroUI when needed (optional):**

npm install @nextui-org/react framer-motion

**Abstraction switcher** (src/components/ui/index.ts):

// Choose a default via env, but allow mixing per-component later.

const lib = process.env.NEXT\_PUBLIC\_COMPONENT\_LIBRARY || 'shadcn';

switch (lib) {

case 'heroui':

export \* from './heroui'; // wrappers re-exported from @nextui-org/react

break;

default:

export \* from './shadcn'; // shadcn/ui exports

}

// Example per-component “best of” mapping (mixed mode)

export { Button } from './heroui/button';

export { Input } from './shadcn/input';

export { Card } from './21st/card';

**HeroUI providers (if used):**

**Cursor rule for centralized imports:**

Always import UI primitives from "@/components/ui" (abstraction layer).

**4) Styling: Tailwind, Tokens & Combo Classes**

* **Design tokens** live in src/lib/tokens.ts (colors, spacing, type, radius).
* **Tailwind config** exposes tokens as CSS vars and adds a few project scales.
* **Combo classes** (globals.css) are reusable patterns (layout, typography, buttons, cards, forms). Use a combo when a utility recipe repeats **3+ times**.

**5) Design Systems (optional flair)**

We can theme for **Glassmorphism / Neumorphism / Brutalism** using component-level class maps and CSS layers. Switch styles via a single config.

**Design system config** (src/lib/design-system.config.ts): map **component→class** per system.

**6) Figma → Cursor (MCP) → Code**

**Setup**

* **Install & configure Figma MCP** in Cursor with Personal Access Token. Add server to ~/.cursor/config.json.
* Put our **generation rules** in .cursorrules.

**Minimal MCP config example** (~/.cursor/config.json):

{

"mcpServers": {

"figma": {

"command": "npx",

"args": ["-y","@figma/mcp-server-figma"],

"env": { "FIGMA\_PERSONAL\_ACCESS\_TOKEN": "YOUR\_FIGMA\_TOKEN" }

}

}

}

**Figma file key + prompts**: use MCP to fetch nodes and generate components with our naming + token rules.

**Project-level Figma mapping** (figma.config.json) to enforce semantic naming & token extraction:

**7) Strapi Integration (headless CMS)**

**Why:** low content churn; we render public pages with ISR and **revalidate on publish**. Editors work in Strapi; the app stays pure React/Next.  
**Where:** src/app/(marketing)/[pathway]/page.tsx etc. fetch via **server actions**/RSC.

**Content model (MVP):**

* Pathway (slug, title, summary, tier: free/premium, order)
* Module (pathway relation, order)
* Lesson (module relation, mdx/html body, assets, estTime, tier)
* Announcement / PromoBanner (optional)

**Fetching pattern (RSC):**

// example server component

import 'server-only';

export async function getPathway(slug: string) {

const url = `${process.env.STRAPI\_URL}/api/pathways?filters[slug][$eq]=${slug}&populate=deep`;

const res = await fetch(url, { headers: { Authorization: `Bearer ${process.env.STRAPI\_TOKEN}` }, next: { revalidate: 3600 }});

return (await res.json()).data?.[0];

}

**ISR + Revalidate on publish:** Strapi → **Webhook** → Vercel /api/revalidate route with the affected slugs.

// /app/api/revalidate/route.ts

import { NextRequest, NextResponse } from 'next/server';

export async function POST(req: NextRequest) {

const sig = req.headers.get('x-strapi-signature');

// verify signature...

const { slug, type } = await req.json();

// tag-based revalidation OR path-based

await fetch(`http://127.0.0.1:3000/api/revalidate?tag=${type}:${slug}`);

return NextResponse.json({ revalidated: true });

}

The low-update nature of your content pairs perfectly with **ISR + targeted revalidation** (our baseline pattern in the Next.js stack).

**8) AARC Proprietary Tools (paid tier)**

Keep the quant endpoints separate from the CMS:

* **Service:** /api/price-options, /api/optimize-portfolio (Node/Python).
* **UI:** generated from Figma as **feature components** under components/features/\* (e.g., OptionPricerPanel, OptimizerForm).
* **Gate:** Next.js middleware checks subscription tier before accessing premium sections.

**9) Development Workflow (Design → Code → CMS → Deploy)**

1. **Design in Figma** with meaningful names (e.g., “Pricing Table”, “Hero Section”).
2. In **Cursor**, run MCP prompts: “Fetch ‘Pricing Table’ frame → generate PricingTable using shadcn + tokens + combo classes (responsive).”
3. **Tokens & combos:** ensure colors/spacing come from tokens.ts; extract repeated patterns into globals.css.
4. **Wire Strapi** data in server components; set revalidate and webhooks for publish.
5. **Deploy to Vercel.**

**Cursor Rules (drop into .cursorrules)**

These consolidate the best of the three guides and tailor them to AARC.

# AARC Webtools — Cursor Rules (Figma→Next.js)

## Core Principles

- Always prefer \*\*semantic naming\*\* over property-based names.

- Abide by the component hierarchy: \*\*ui → shared → features → sections → layout\*\*.

- Use \*\*TypeScript\*\* with explicit prop interfaces.

- Use \*\*design tokens\*\* from `src/lib/tokens.ts` (no hardcoded colors/spacing).

- Use \*\*combo classes\*\* from `src/styles/globals.css` for repeated patterns (extract when a pattern repeats 3+ times).

## Imports & Libraries

- Import UI primitives from the \*\*abstraction layer\*\*: `import { Button, Card, Input } from '@/components/ui'`.

- The underlying library (shadcn or HeroUI) is selected in `components/ui/index.ts`. Do not import directly from libraries.

## Figma MCP Workflow

- Before generation, \*\*fetch latest\*\* design data via the \*\*Figma MCP server\*\*.

- Translate Figma layers to \*\*semantic components\*\*:

- Examples: `HeroSection`, `PricingTable`, `NavigationMenu`, `OptionPricerPanel`.

- Ignore generic names (Frame/Group/Rectangle). Use context/purpose.

- Extract colors/spacing/typography to \*\*`tokens.ts`\*\*; prefer \*\*HSL\*\* and CSS vars.

- Map Auto Layout → Tailwind flex/grid; padding/gap → `p-\*/gap-\*`; type styles → typography tokens.

## Component Structure

- Files in `kebab-case` (e.g., `hero-section.tsx`); components in `PascalCase`.

- Place components by intent:

- UI primitives → `components/ui/`

- Sections → `components/sections/`

- Features (domain) → `components/features/`

- Layout → `components/layout/`

- Shared → `components/shared/`

- Use \*\*shadcn\*\* or \*\*HeroUI\*\* components before creating customs.

- Include \*\*responsive\*\* classes: base → mobile; `md:`/`lg:` for larger breakpoints.

## Styling Priority

1) \*\*Design Tokens\*\* (`tokens.ts`)

2) \*\*Combo Classes\*\* (`globals.css`)

3) \*\*Tailwind utilities\*\* (one-offs)

4) \*\*Component props\*\* (e.g., `variant`, `size`, `color`)

## Example Generation Prompt (Good)

“Generate \*\*PricingTable\*\* from Figma frame ‘Pricing — Table v2’:

- Import primitives from `@/components/ui`

- Use tokens for colors/spacing/typography

- Extract repeated patterns into combo classes if used 3+ times

- Type all props (plan name, price, features[])

- Responsive: 1 col mobile, 2 col tablet, 3 col desktop

- Accessibility: semantic table/aria labels”

## Class Naming

- Use \*\*semantic\*\* class names scoped by component:

- `pricing-table`, `pricing-card`, `pricing-feature-list`, `hero-section`, `nav-menu`, `option-pricer-form`

- Avoid property-based names (`flex-col-center`, `blue-button`, `grid3x3`).

## Strapi Integration Hooks

- When a component is content-driven, add \*\*data props\*\* shaped like Strapi responses.

- Avoid CMS-specific logic in UI; fetch data in \*\*server components\*\* or \*\*loaders\*\*.

- For publish events, emit a \*\*revalidation webhook\*\* to Vercel (`/api/revalidate`).

## Accessibility & Performance

- Use semantic HTML (`nav/main/section/article`), labels for icon buttons, alt text, focus order.

- Use `next/image`, loading/error states, and avoid client components unless needed.

## Checks (must pass)

- [ ] Semantic component & class names

- [ ] Tokens used (no hardcoded hex/rgb)

- [ ] Combo classes used where repeated

- [ ] shadcn/HeroUI primitives leveraged

- [ ] Responsive breakpoints included

- [ ] TypeScript props defined

- [ ] A11y attributes present

(Sources: Figma MCP guide, multi-library abstraction, naming/tokens/combo-classes standards.)

**Quick Start Commands**

**Next + Tailwind + shadcn base:**

npx create-next-app@latest aarc-webtools --typescript --tailwind --app

npx shadcn-ui@latest init

npx shadcn-ui@latest add button card input

npm i class-variance-authority clsx tailwind-merge -D @types/node

**HeroUI optional:**

npm i @nextui-org/react framer-motion

**Figma MCP server in Cursor:**

{

"mcpServers": {

"figma": { "command":"npx","args":["-y","@figma/mcp-server-figma"], "env":{"FIGMA\_PERSONAL\_ACCESS\_TOKEN":"..."} }

}

}

**Design-system toggle (optional):** update design-system.config.ts + import the matching CSS in globals.css.

**What you’ll get from this setup**

* Figma components → **typed** Next.js components using our abstraction, tokens, and combos (auto-generated via MCP).
* Ability to **switch** between **shadcn** and **HeroUI** without rewriting code.
* Clean, maintainable style via **tokens + combo classes**; no property-based naming.
* Strapi-driven content with **ISR + webhook-based revalidation** suitable for low-churn education paths.

If you want, I can also drop in a minimal **Strapi content schema** (JSON) + a ready-to-paste **/api/revalidate** route using tag-based invalidation next.

Awesome — here are both deliverables, ready to paste into your repos.

**1) Strapi content schemas (collection types)**

Place these files in your Strapi project under the indicated paths (Strapi v4/v5 style). After adding, run a build so Strapi migrates the DB.

Root: ./strapi/ (or wherever your Strapi app lives)  
Content types live at: strapi/src/api/<type>/content-types/<type>/schema.json

**strapi/src/api/pathway/content-types/pathway/schema.json**

{

"collectionName": "pathways",

"info": {

"singularName": "pathway",

"pluralName": "pathways",

"displayName": "Pathway",

"description": "Top-level learning track"

},

"options": {

"draftAndPublish": true

},

"attributes": {

"title": {

"type": "string",

"required": true

},

"slug": {

"type": "uid",

"targetField": "title",

"required": true,

"unique": true

},

"summary": {

"type": "text"

},

"order": {

"type": "integer",

"default": 1

},

"tier": {

"type": "enumeration",

"enum": ["free", "premium"],

"default": "free",

"required": true

},

"modules": {

"type": "relation",

"relation": "oneToMany",

"target": "api::module.module",

"mappedBy": "pathway"

},

"seoTitle": {

"type": "string"

},

"seoDescription": {

"type": "string"

},

"ogImage": {

"type": "media",

"multiple": false

}

}

}

**strapi/src/api/module/content-types/module/schema.json**

{

"collectionName": "modules",

"info": {

"singularName": "module",

"pluralName": "modules",

"displayName": "Module",

"description": "Module within a Pathway"

},

"options": {

"draftAndPublish": true

},

"attributes": {

"title": {

"type": "string",

"required": true

},

"slug": {

"type": "uid",

"targetField": "title",

"required": true,

"unique": true

},

"order": {

"type": "integer",

"default": 1

},

"pathway": {

"type": "relation",

"relation": "manyToOne",

"target": "api::pathway.pathway",

"inversedBy": "modules",

"required": true

},

"lessons": {

"type": "relation",

"relation": "oneToMany",

"target": "api::lesson.lesson",

"mappedBy": "module"

}

}

}

**strapi/src/api/lesson/content-types/lesson/schema.json**

{

"collectionName": "lessons",

"info": {

"singularName": "lesson",

"pluralName": "lessons",

"displayName": "Lesson",

"description": "Lesson within a Module"

},

"options": {

"draftAndPublish": true

},

"attributes": {

"title": {

"type": "string",

"required": true

},

"slug": {

"type": "uid",

"targetField": "title",

"required": true,

"unique": true

},

"module": {

"type": "relation",

"relation": "manyToOne",

"target": "api::module.module",

"inversedBy": "lessons",

"required": true

},

"tier": {

"type": "enumeration",

"enum": ["free", "premium"],

"default": "free",

"required": true

},

"estimatedTimeMins": {

"type": "integer",

"default": 5

},

"body": {

"type": "richtext"

},

"assets": {

"type": "media",

"multiple": true

},

"seoTitle": {

"type": "string"

},

"seoDescription": {

"type": "string"

},

"ogImage": {

"type": "media",

"multiple": false

}

}

}

**strapi/src/api/announcement/content-types/announcement/schema.json**

{

"collectionName": "announcements",

"info": {

"singularName": "announcement",

"pluralName": "announcements",

"displayName": "Announcement",

"description": "Site-wide announcements or promos"

},

"options": {

"draftAndPublish": true

},

"attributes": {

"title": {

"type": "string",

"required": true

},

"body": {

"type": "richtext"

},

"active": {

"type": "boolean",

"default": true

},

"startsAt": {

"type": "datetime"

},

"endsAt": {

"type": "datetime"

},

"tier": {

"type": "enumeration",

"enum": ["all", "free", "premium"],

"default": "all",

"required": true

}

}

}

Notes

* Slugs are unique to simplify tag-based cache keys.
* draftAndPublish: true lets you preview and only trigger revalidation on publish.
* If you want Markdown instead of rich text, change body to "type": "text" and store MD.

**2) Next.js tag-based revalidation route**

This route verifies the Strapi webhook (HMAC-SHA256), figures out which tags to refresh, and calls revalidateTag for each. It supports single-item and “global” refreshes.

Path (App Router): app/api/revalidate/route.ts  
Env vars needed in Vercel:

* STRAPI\_WEBHOOK\_SECRET — shared secret you put in the Strapi webhook
* (optional) ALLOWED\_EVENT\_SOURCE — if you want to restrict by host

// app/api/revalidate/route.ts

import { NextRequest, NextResponse } from 'next/server';

import { revalidateTag } from 'next/cache';

import crypto from 'crypto';

export const runtime = 'nodejs'; // ensure Node runtime for crypto

type StrapiWebhook = {

event: string; // e.g., "entry.publish"

model?: string; // v5 may include model/uid

uid?: string; // e.g., "api::lesson.lesson"

entry?: {

id: number | string;

slug?: string;

// other fields available if you included them in the webhook payload

};

};

function timingSafeEqual(a: string, b: string) {

const aBuf = Buffer.from(a);

const bBuf = Buffer.from(b);

if (aBuf.length !== bBuf.length) return false;

return crypto.timingSafeEqual(aBuf, bBuf);

}

async function verifySignature(req: NextRequest, rawBody: string) {

const secret = process.env.STRAPI\_WEBHOOK\_SECRET || '';

const signature = req.headers.get('x-strapi-signature') || '';

if (!secret || !signature) return false;

const hmac = crypto.createHmac('sha256', secret).update(rawBody).digest('hex');

return timingSafeEqual(hmac, signature);

}

function computeTags(payload: StrapiWebhook) {

// Normalise uid/model → simple type key

const uid = payload.uid || payload.model || '';

const type =

uid.includes('pathway') ? 'pathway' :

uid.includes('module') ? 'module' :

uid.includes('lesson') ? 'lesson' :

uid.includes('announcement') ? 'announcement' : 'unknown';

const slug = payload.entry?.slug;

const tags = new Set<string>([

'site:all', // global fallback

`type:${type}` // all pages of this type

]);

if (type === 'pathway' && slug) {

tags.add(`pathway:${slug}`);

// modules & lessons under this pathway often appear on the pathway page

tags.add(`list:modules:${slug}`);

tags.add(`list:lessons:${slug}`);

}

if (type === 'module' && slug) {

tags.add(`module:${slug}`);

// module pages list lessons

tags.add(`list:lessons:${slug}`);

}

if (type === 'lesson' && slug) {

tags.add(`lesson:${slug}`);

}

if (type === 'announcement') {

tags.add('site:announcements');

}

return Array.from(tags);

}

export async function POST(req: NextRequest) {

try {

// We need the raw body for signature verification:

const raw = await req.text();

const ok = await verifySignature(req, raw);

if (!ok) {

return NextResponse.json({ error: 'Invalid signature' }, { status: 401 });

}

const payload = JSON.parse(raw) as StrapiWebhook;

// Optional: restrict by host

const source = req.headers.get('host') || '';

const allowed = process.env.ALLOWED\_EVENT\_SOURCE;

if (allowed && !source.endsWith(allowed)) {

return NextResponse.json({ error: 'Forbidden source' }, { status: 403 });

}

const tags = computeTags(payload);

for (const tag of tags) {

revalidateTag(tag);

}

return NextResponse.json({ revalidated: true, tags });

} catch (e: any) {

return NextResponse.json({ error: e?.message || 'Unknown error' }, { status: 500 });

}

}

**How to use tags in your data fetching**

When you fetch content in Server Components or Route Handlers, attach **stable tags** so the above route can invalidate them precisely.

**Examples:**

// Fetch a pathway page (RSC or route handler)

export async function getPathway(slug: string) {

const res = await fetch(

`${process.env.STRAPI\_URL}/api/pathways?filters[slug][$eq]=${slug}&populate=deep`,

{ next: { tags: ['site:all', 'type:pathway', `pathway:${slug}`] } }

);

const json = await res.json();

return json?.data?.[0];

}

// Fetch module list for a pathway card

export async function getModulesForPathway(pathwaySlug: string) {

const res = await fetch(

`${process.env.STRAPI\_URL}/api/modules?filters[pathway][slug][$eq]=${pathwaySlug}&sort=order:asc`,

{ next: { tags: ['site:all', 'type:module', `list:modules:${pathwaySlug}`] } }

);

return (await res.json())?.data;

}

// Fetch a single lesson page

export async function getLesson(slug: string) {

const res = await fetch(

`${process.env.STRAPI\_URL}/api/lessons?filters[slug][$eq]=${slug}&populate=deep`,

{ next: { tags: ['site:all', 'type:lesson', `lesson:${slug}`] } }

);

return (await res.json())?.data?.[0];

}

// Announcements banner

export async function getActiveAnnouncements() {

const nowIso = new Date().toISOString();

const res = await fetch(

`${process.env.STRAPI\_URL}/api/announcements?filters[$and][0][active][$eq]=true&filters[$and][1][startsAt][$lte]=${nowIso}&filters[$and][2][endsAt][$gte]=${nowIso}`,

{ next: { tags: ['site:all', 'type:announcement', 'site:announcements'] } }

);

return (await res.json())?.data;

}

**Strapi webhook (one-time setup)**

* Create a **Webhook** in Strapi Admin → Settings → Webhooks.
* **URL**: https://<your-vercel-domain>/api/revalidate
* **Headers**: add x-strapi-signature: {{ generated by Strapi secret }} (Strapi signs body with your secret)
* **Secret**: set to your STRAPI\_WEBHOOK\_SECRET (also set the same value in Vercel env).
* Trigger on events: entry.publish, entry.unpublish, entry.update for the four content types.

Tip: include the slug field in the webhook payload so computeTags can target item-level tags right away.

**Sanity checks**

* After adding schemas, run yarn build && yarn develop (or npm run ...) in Strapi; create one Pathway/Module/Lesson to confirm slugs and relations.
* Deploy the Next.js route; set STRAPI\_WEBHOOK\_SECRET in Vercel.
* Create/publish an entry in Strapi → confirm the route returns { revalidated: true, tags: [...] }.
* Visit an affected page and ensure content updates without a full redeploy.

If you want, I can also add a tiny **preview** route (draft preview via token) and a **sample Strapi webhook payload template** tailored to your instance.