STRETCHLEARN · WORKBOOK

Design Systems in Figma — Workbook

This workbook turns the course into the audits, decisions, and specs you need to build a real Figma design system. Work each section as you go: structure your files and component inventory, design a two-tier token system with light and dark modes, plan your component properties and auto-layout behavior, then write the documentation and governance that decide whether the system is adopted. The templates are built to be filled in for your own product and reused as the working documents you take into Figma and into team reviews.

Foundations and File Setup

Decide what belongs in the system, structure the library file and pages, and inventory every component before you build.

Worksheet: Component Inventory and Layer Classification

List every recurring UI element across your product, then classify each one into the system layer it belongs to and decide whether it earns a place in the library. Do this before building anything; it is the scope plan the rest of the system follows.

UI element (e.g., primary button, text input, card, navbar, empty state)

How many screens or places does it appear? (count or estimate)

System layer (foundation / component / pattern / one-off asset)

Atomic level (atom / molecule / organism / template)

Belongs in the library? (yes / no — one-offs stay out)

Existing duplicates today (how many near-identical versions exist)

Owner / who builds it

Build priority (1 = blocking, 2 = soon, 3 = later)

Exercise: Structure Your Library File

Set up your dedicated design-system file (separate from product files) with a clear page structure, then justify each choice against findability and performance.

• Is the system in its own file and project, separate from product work-in-progress, and how will consuming teams reach it?

• What pages did you create (Cover, Getting started, Foundations, Components, Documentation, Playground), and how does a newcomer find the button in under ten seconds?

• How will you handle safe changes — native branching, or a duplicate-review-merge process — and who is allowed to publish?

• At what point would you split into multiple library files (foundations, components, icons), and what performance signal would trigger that?

Checklist: Setup Readiness Check

[ ] I separated design system (the product) from UI kit (the file) and style guide (the rules) in my own plan
[ ] The library lives in its own dedicated Figma file, not mixed with product screens
[ ] Pages are named and ordered so the system is findable in seconds
[ ] I inventoried recurring elements and classified each into a layer before building
[ ] One-off assets are deliberately kept out of the library
[ ] I have a defined safe-change process so nobody edits the live library casually

Tokens, Variables, and Theming

Build a two-tier token system with primitives and semantics, then theme it for light, dark, and any brands using modes.

Worksheet: Two-Tier Token Map

Define your semantic tokens and the primitive each one aliases in every mode. Fill one row per semantic token. Components will bind only to the semantic column, never to primitives.

Semantic token (e.g., color/action/primary, color/bg/surface, color/text/danger)

Purpose / where it is used

Primitive alias in Light mode (e.g., color/blue/600)

Primitive alias in Dark mode (e.g., color/blue/400)

Resolved hex Light

Resolved hex Dark

Contrast vs. its background, Light (ratio + pass/fail AA)

Contrast vs. its background, Dark (ratio + pass/fail AA)

Exercise: Design Your Scales

Define the numeric foundations of the system — spacing, type, and radius — as primitive tokens with a consistent step, then write them out so layouts stay rhythmic.

• What spacing base and scale did you choose (e.g., 4px base: 4, 8, 12, 16, 24, 32, 48, 64), and how many steps do you actually need?

• What type scale and ratio did you pick (e.g., 16px base at a 1.25 major-third ratio rounded to clean numbers), and what are the resulting sizes?

• What radius and elevation tokens did you define, and how do elevation overlays read in dark mode without using pure black (target around #121212)?

• Which values in your current designs are hard-coded today, and what is your plan to bind them to variables instead?

Checklist: Token System Check

[ ] Primitives describe what a value is; semantics describe what it is for
[ ] Components bind only to semantic tokens, never to raw primitives or hex values
[ ] Primitives and Semantic live in separate collections, with semantics aliasing primitives
[ ] Modes carry the theming so light and dark resolve from one token set
[ ] Dark mode is its own considered theme, not an inversion of light
[ ] Every semantic text-on-surface pair passes WCAG AA (4.5:1 normal, 3:1 large) in both modes

Components and Auto Layout

Plan each component's properties, variants, and responsive auto-layout behavior so one component covers many states.

Worksheet: Component Property Specification

For each core component, specify its properties so it collapses many states into one configurable component. Decide deliberately what is a variant (mutually exclusive) versus a boolean (independent toggle).
Component name (e.g., Button, Input, Card)

Variant properties + values (e.g., Type: primary/secondary/ghost/danger; Size: sm/md/lg)

Boolean properties (e.g., Has leading icon, Has trailing icon)

Text properties exposed (e.g., Label)

Instance-swap slots (e.g., Icon, Media)

State variant values (default / hover / focus / disabled)

Default variant (the most-used configuration)

Estimated variant count, and how booleans/slots kept it from exploding

Exercise: Auto-Layout Stress Test

Take one real component and define its auto-layout behavior, then test it against extreme content. A system component must survive one word and a paragraph.
• What is the auto-layout direction, gap, and padding, and which spacing tokens are they bound to?

• For each child, is it set to Hug, Fill, or Fixed, and why (e.g., label hugs, full-width button fills, icon fixed)?

• Did you set min and max width or height so the component does not stretch forever or fail to wrap?

• When you tested with a one-word label and a three-line label, what broke, and how did you fix it?

Checklist: Component Architecture Check

[ ] Mutually exclusive options are variants; independent toggles are booleans
[ ] Interaction states live in one State variant, not as separate components
[ ] Every component is wrapped in auto layout and tested with short and long content
[ ] Larger components nest real instances of smaller ones, never flattened copies
[ ] Slots (instance-swap) cover content that varies widely, with sensible defaults
[ ] Variant property and value names match the code's prop values where possible

Publishing, Documentation, and Governance

Publish and version the library, document each component for correct use, and define the contribution and adoption workflow.

Worksheet: Component Documentation Page

Draft the documentation for one component so a stranger on another team could use it correctly from the docs alone. Fill this out per component as part of its definition of done.

Component name

Purpose (1-2 sentences: what it is for, what problem it solves)

Anatomy (labeled parts: container, label, leading icon, helper text, etc.)

Variants and properties (each option and what it is for)

Do example (a correct use)

Do-not example (a wrong use to avoid)

Behavior and accessibility (states, focus/keyboard, min touch target 44x44pt or 48x48dp, contrast)

Content guidance (label wording, length, capitalization, tone)

Exercise: Choose Your Governance Model and Release Plan

Decide who can change the system and how updates ship, then write it down so the system has a clear, low-friction path from need to published component.

• Which operating model fits you — solitary, centralized, or federated (per Nathan Curtis) — and who owns the system as a product?

• What is your contribution workflow (request, triage, build, review, publish), and through what channel do people submit requests?

• How will you handle breaking vs. non-breaking changes, batching, and deprecation runways so teams are not surprised?

• Which adoption metrics will you track (component insertions, detach rate, coverage, support load) and where do they come from?

Worksheet: Release Changelog Entry

Practice writing one release entry the way consumers will read it before accepting an update. Fill one block per release.

Version / date

Added (new components, variants, tokens)

Changed (token values, property tweaks — non-breaking)

Breaking changes (renames, removals, restructured properties)

Deprecated (what is being phased out and its replacement)

Migration notes (what consumers must do, and the migration window)

Checklist: Launch and Governance Check

[ ] The library is published from a controlled file with batched, documented releases
[ ] Breaking changes are announced, scheduled, and given a migration path
[ ] Deprecated components are marked and given a runway before deletion
[ ] Every shipped component has documentation as part of its definition of done
[ ] A written contribution guide covers naming, token rules, the build checklist, and how to request changes
[ ] Adoption is measured (insertions, detach rate, coverage), and detaches are treated as backlog items, not user error

Your Action Plan

1. Create a dedicated design-system Figma file with Cover, Getting started, Foundations, Components, Documentation, and Playground pages.
2. Inventory every recurring UI element, classify each by layer, and keep one-off assets out of the library.
3. Build the Primitives collection: color, spacing (4px base), type (modular ratio), radius, and elevation tokens with no modes.
4. Build the Semantic collection as aliases into Primitives, and add Light and Dark modes that resolve correctly and pass WCAG AA.
5. Build base components (button, input, icon, badge, avatar) with deliberate variant and boolean properties, all bound to tokens.
6. Wrap every component in auto layout, set Hug/Fill/Fixed resizing, and stress-test each with one word and a paragraph.
7. Compose larger components by nesting real instances and using instance-swap slots for variable content.
8. Publish the library, write a Getting started page, and document each component with anatomy plus do and do-not examples.
9. Write a contribution guide and pick a governance model (solitary, centralized, or federated) with a request-to-release workflow.
10. Turn on library analytics, track insertions and detach rate, review gaps monthly, and ship improvements as planned releases.