STRETCHLEARN · WORKBOOK

Design Tokens — Workbook

This workbook turns the course into a built token set you can actually ship. You will inventory raw values into globals, name and tier them, theme them for light and dark, and plan the export and sync from a single source. Work through one section per module, then use the templates to capture your real tokens.

Tiers and Taxonomy

Sort your existing visual decisions into global, alias, and component tiers and spot where you are binding too low.

Exercise: Hex hunt: find your duplicated values

Open one screen of your current product or design file. List every distinct color, spacing, radius, and font size you can find, then mark which ones repeat. Repetition is the strongest signal that a value wants to be a token.

• Which three values appear most often across the screen, and how many times each?

• Which colors are nearly-but-not-exactly equal (for example two slightly different grays) and should collapse into one token?

• Where is a literal hex or pixel value written directly on a component instead of referencing a shared value?

• If the brand color changed tomorrow, how many separate places would you have to edit by hand right now?

Worksheet: Tier sort for ten values

Pick ten values from your hex hunt. For each, fill the row: write a global name for the raw value, an alias name for its role, and a component name only if it is truly component-specific. Leave the component column blank when an alias is enough.

Raw value (hex / px / number)

Global token name (e.g. color/blue/600)

Alias token name (e.g. color/action/primary)

Component token name (only if needed)

What is it for? (role in one phrase)

Tier a component should bind to

Checklist: Tier sanity check

[ ] Every raw hex or pixel value has a global token name
[ ] No alias or component token holds a literal value; each references a global
[ ] Each value is bound to the highest tier that still carries its meaning
[ ] Component tokens exist only where a value genuinely differs from a shared alias
[ ] I can name the one token to edit for a full rebrand (the primary action alias)

Naming and Structure

Lock a naming vocabulary and express a slice of your tokens in the DTCG JSON shape with types and references.

Exercise: Build your closed vocabulary

Write the approved word for each concept and ban the synonyms. This single page prevents synonym drift, where surface, bg, and background all mean the same thing. Decide each one now and commit.

• Background fill: which one word do you approve (surface, bg, or background) and which are banned?

• Spacing: do you use a t-shirt scale (xs to xl) or a numeric ramp (1 to 12), and what is the base unit in px?

• Color shades: do you use a numeric ramp (50 to 950) or named steps, and is it consistent across every color?

• State modifiers: which words are allowed for states (e.g. hover, active, disabled) and what is the order in a name?

Worksheet: DTCG token spec for one color and one spacing token

Express two tokens in the standard W3C format using plain fields. Do one global color, one alias color that references it, and one spacing token. Note the type on each and use a reference (the global path in braces) for the alias value.

Token 1 group path (e.g. color > blue)

Token 1 $type (color)

Token 1 $value (hex)

Alias group path (e.g. color > action)

Alias $type and $value (reference to the global path)

Spacing token path, $type (dimension), and $value (e.g. 16px)

One $description per token explaining intent

Worksheet: File and collection layout plan

Decide where each tier lives and where theme variation goes. Keep core mode-free and put per-theme values on the semantic tier.

File for globals (e.g. tokens/core.json)

File for aliases (e.g. tokens/semantic.json)

Component tokens: separate file or none?

Where theme variation lives (which tier)

Figma collections that mirror these files

Rule (one sentence): core never knows about themes

Checklist: Naming and structure check

[ ] Names read general to specific (category, property, modifier)
[ ] One approved word per concept; synonyms are written down and banned
[ ] Globals describe the value; aliases describe the role; no value names baked into aliases
[ ] Each token has an explicit $type, set on the group where possible
[ ] Aliases reference globals rather than copying values
[ ] Globals are mode-free; all theme variation lives on the semantic tier

Theming and Accessibility

Define light and dark (and optionally a brand) theme by rebinding semantics, and verify contrast at the token level.

Exercise: Two-mode rebind walkthrough

Take three semantic roles and resolve them for both light and dark by pointing at globals, not by creating new tokens. Confirm a single component bound to these roles would invert correctly.

• color/bg/surface: which global in light mode, which in dark mode?

• color/text/default: which global in light mode, which in dark mode?

• color/action/primary: does it stay the same across modes or shift, and why?

• Did you avoid creating any -light or -dark suffixed tokens? If not, which one should become a mode value instead?

Worksheet: Contrast audit per theme

For each critical pair, record the contrast ratio in light and dark and whether it clears the WCAG 2.1 AA target (4.5 to 1 for normal text, 3 to 1 for large text and UI boundaries). Fix any failing pair by adjusting the global it references.

Pair (foreground role on background role)

Ratio in light mode

Pass AA light? (yes/no, target used)

Ratio in dark mode

Pass AA dark? (yes/no, target used)

Fix if failing (which global to change)

Checklist: Theme and accessibility check

[ ] Only the semantic tier carries modes; the core palette is mode-free
[ ] No duplicated per-mode tokens; one name resolves per mode
[ ] text/default on bg/surface clears 4.5 to 1 in both light and dark
[ ] action/primary versus its on-action text clears 4.5 to 1
[ ] Border and focus-ring roles clear 3 to 1 against adjacent surfaces
[ ] A min touch-target token (44px iOS / 48px Android) exists and binds to interactive heights

Export, Sync, and Governance

Plan the build from one source to CSS and JSON, pick a source of truth for Figma-code sync, and set up versioning and ownership.

Exercise: Choose your source of truth and sync loop

Decide which side owns the truth and is allowed to edit tokens, and make the other strictly downstream. Then describe the exact change loop so no two truths drift apart.

• Is Figma (via Tokens Studio) or the repo JSON your source of truth, and who edits it?

• What is the path for a token change (e.g. edit, push branch, open pull request, review, merge)?

• What runs in continuous integration to catch a broken reference or bad type before merge?

• How often does the downstream side pull so it never drifts more than one change behind?

Worksheet: Style Dictionary platform plan

List the outputs you need from the single token source. For each platform, note the output file and what the build must do (resolve references, convert units, transform names).

Platform (e.g. css, json, scss, ios)

Output file path

Naming transform (e.g. slash path to kebab-case CSS variable)

Unit transform needed (e.g. px to rem)

Consumer (who imports this output)

Worksheet: Version and deprecation entry

Draft one changelog entry for an upcoming token change and classify the version bump. Practice deprecating rather than deleting.

Token(s) changed

Change type (patch value / minor add / major rename or remove)

New version number

If removing: replacement token and deprecation note

Grace period (how many releases before delete)

Migration note for consumers

Checklist: Export and governance check

[ ] One token source produces every platform output via the build, no hand-copied values
[ ] A single named source of truth exists for Figma-code sync
[ ] Every token change goes through a pull request and is recorded
[ ] The build runs in continuous integration and fails on broken references or bad types
[ ] The token set is versioned with semantic versioning and a changelog
[ ] Tokens are deprecated with a replacement before deletion, and an owner is named

Your Action Plan

1. Run a hex hunt on one real screen and list every distinct color, spacing, radius, and font size
2. Sort those values into global, alias, and optional component tiers using the tier-sort worksheet
3. Write a closed naming vocabulary: one approved word per concept, synonyms banned
4. Express your starter set in DTCG JSON: globals with values, aliases referencing them, explicit types
5. Lay out files so core is mode-free and all theme variation lives on the semantic tier
6. Build light and dark by rebinding semantics, creating no per-mode duplicate tokens
7. Audit contrast for every critical pair in both modes against WCAG 2.1 AA and fix failing pairs
8. Add a Style Dictionary build that emits CSS custom properties and a JSON theme from the one source
9. Pick a source of truth (Figma via Tokens Studio or the repo) and route all changes through pull requests
10. Version the set with semantic versioning, deprecate before deleting, and name an owner