### Prompts for Windsurf/Cursor (English, referencing SPEC.md with v4 and new features)

Below are actionable, copy-paste prompts you can use in Windsurf/Cursor. Each prompt references SPEC.md as the source of truth and targets a concrete milestone toward Tailwind v4 tokens, deterministic class ordering, richer matchers, and better reporting/CLI.

#### 1) Add Theme Tokens Loader (v4-first with v3 fallback)

Goal: Implement loadTokens that prefers CSS custom properties (v4) and falls back to v3 config. Update MatchCtx to carry version, tokens, opts.screens.

Prompt: ~~- You are working in the css-windify monorepo. Use SPEC.md as the single source of truth, especially sections: “Public APIs → Theme and token handling,” “Tailwind v4 Migration Plan → Tokens loader (new), Version detection.” - Implement a new tokens loader in css-windify-core: - Function: loadTokens(options?: { cssPath?: string; configPath?: string; }): Promise<ThemeTokens> - Behavior: Try to read CSS custom properties from cssPath (parse :root or layers). Extract tokens for spacing, fontSize, lineHeight, colors, and screens. When not found, fallback to v3 theme by delegating to loadTheme(configPath) and project config into ThemeTokens. - Add ThemeTokens type and the v4-first detection logic. The output must include version, source, and numeric screens. - Update MatchCtx and TailwindifyOptions per SPEC.md (version: ‘auto’ | ‘v3’ | ‘v4’, screens override). - Add unit tests for: (1) CSS-only tokens, (2) v3 config fallback, (3) missing CSS file gracefully handled.~~

~~Acceptance criteria: - loadTokens compiles, exported by core index, covered by tests. - ThemeTokens aligns with SPEC.md structure. - version: 'auto' resolves to ‘v4’ when CSS tokens present, otherwise ‘v3’, with a v3-fallback warning path available for resolvers.~~

#### 2) Update Resolvers to Prefer v4 Tokens and Support Approximation

Goal: Switch resolvers (resolveSpacingToken, resolveFontSizeToken, resolveLineHeightToken, resolveColorToken) to use MatchCtx.tokens (v4) first; fallback to v3. Implement resolveNearestTokenPx and thresholds.

Prompt: ~~- Follow SPEC.md sections “Resolvers,” “Thresholds by category,” and “Tailwind v4 Migration Plan → Resolvers in v4.” - Implement resolveNearestTokenPx(valuePx, tokenMapPx) in core utils. - Update spacing, font-size, line-height resolvers to: - If ctx.version === ‘v4’ and tokens available, prefer tokens. - If opts.approximate and within threshold (opts.thresholds), choose nearest token and emit approximate warning with {prop, original, tokenKey, diffPx} metadata. - If opts.strict and no exact token, return arbitrary and emit token-miss warning. - Else fallback to v3 theme (emit v3-fallback if version was ‘v4’). - Update color resolver to prefer tokens when possible; otherwise behave as before.~~

~~Acceptance criteria: - Resolvers select v4 tokens when available; tests verify strict vs approximate and v3 fallback. - Structured warnings for approximate, token-miss, v3-fallback.~~

#### 3) Deterministic Class Ordering Implementation

Goal: Replace simple dedupe with deterministic ordering: by variantPrefix, groupOrder, classKey.

Prompt: ~~- Implement the “Class Ordering” algorithm exactly as in SPEC.md: groups, prefix-based inference, and sorting tuple (variantPrefix, groupOrder, classKey, classFull). Keep stable and deduped results. - Add a getClassGroup(className) helper and a sortClasses(classes: string[]) that applies the algorithm. - Ensure variant prefixes (e.g., md:hover:) are part of the key and sorted lexicographically. - Write unit tests that verify ordering for mixed classes with variants (responsive + pseudo).~~

~~Acceptance criteria: - Sorting is deterministic, tested with complex examples. - Dedupe is preserved.~~

#### 4) Expand Matchers Coverage (Flex/Grid, Backgrounds, Borders, Typography)

Goal: Implement the new mapping coverage defined in SPEC.md.

Prompt: - Based on SPEC.md sections “Matching and Mapping Rules (v3 baseline)” and sub-sections you added (Borders: style; Flex/Grid: basis/grow/shrink, grid-auto-flow, placements; Backgrounds: repeat/attachment/origin/clip; Typography: letter-spacing and text-decoration). - Add or extend handlers: - flex-basis → spacing tokens or basis-[value] - flex-grow/shrink → grow, grow-0, shrink, shrink-0 - grid-auto-flow → grid-flow-row|col|dense|row-dense|col-dense - grid placements → col-span-\*, col-start-\*, col-end-\*, row-\* - background-repeat → bg-no-repeat|repeat|repeat-x|repeat-y|round|space - background-attachment → bg-fixed|local|scroll - background-origin/clip → arbitrary fallback - border-style → border-solid|dashed|dotted|double, otherwise arbitrary property - letter-spacing → tokens or tracking-[value] - text-decoration → underline, line-through, no-underline; decoration-\* → arbitrary when needed - Add tests for each new handler with strict and approximate (where applicable).

Acceptance criteria: - Each property maps per SPEC.md with appropriate fallbacks and warnings. - Tests pass and demonstrate coverage.

#### 5) Shorthand Parsing Improvements (background, border, font, transition, animation)

Goal: Robust decomposition of complex shorthands and fallback to arbitrary when needed.

Prompt: - Implement or enhance parsers to decompose: - background: color, image, position, size, repeat, attachment - border: width, style, color - font: style/variant/weight/size/line-height/family → map to utilities where straightforward; arbitrary for the rest - transition: transition, duration-\*, ease-\*, delay-\* when determinable; arbitrary otherwise - animation: support basic animate-none else arbitrary [animation:...] - Use SPEC.md “Shorthand handling” and related mapping sections as acceptance criteria. - Add unit tests per shorthand type with representative cases.

Acceptance criteria: - Shorthands decomposed into correct utilities or safe arbitrary fallbacks with warnings. - No regressions on existing shorthand logic.

#### 6) Variants Expansion (visited, focus-visible, focus-within, odd/even, group/peer)

Goal: Extend withVariant(s) to support more pseudo-class/state variants and ensure correct chaining and dedupe.

Prompt: - Extend variants system to cover: visited, focus-visible, focus-within, first, last, odd, even, group-(hover/focus/active/etc), peer-(focus/checked/etc). - Preserve order and hierarchy per SPEC.md; ensure chaining works like md:dark:group-hover:focus:.... - Dedupe repeated variants while preserving intended order. - Add tests that combine responsive and pseudo variants with multiple classes.

Acceptance criteria: - Variants apply in the order provided and compose correctly across classes. - Tests confirm the output matches SPEC.md examples and rules.

#### 7) Reporter Enhancements (structured warnings and stats)

Goal: Enrich summarize to return both text and a structured stats object.

Prompt: - Implement the summarize return signature from SPEC.md (text + stats). Include: - totals (matched, total, percentage, nonArbitrary) - byCategory coverage - warningsByCategory counts - samples (classes, warnings) - Ensure warning types include the new categories (token-miss, v3-fallback) and structured metadata for approximate. - Add tests for summarize using synthetic TransformResult arrays.

Acceptance criteria: - Reporter returns a meaningful text summary and a structured stats object. - Tests validate counts and sample extraction.

#### 8) CLI MVP

Goal: Implement an initial CLI that can process files/stdin and output JSON or Markdown summaries.

Prompt: - Implement css-windify-cli per SPEC.md “CLI implementation roadmap (Basic)”. - Features: - Input: CSS from file(s) or stdin. - Options: --strict, --approximate, --thresholds.spacing=2, --version=auto|v3|v4, --screens=sm:640,md:768,..., --report=json|markdown - Output: For each selector, ordered class string, plus a summary via summarize. - Keep dependencies minimal; use core package APIs. - Add basic integration tests (can be script-level) verifying JSON and Markdown outputs.

Acceptance criteria: - CLI builds and runs locally. - Produces correct ordered classes and a summary report. - Flags correctly influence matching behavior.

#### 9) Integration Tests for v4 Tokens vs v3 Theme Parity

Goal: Ensure consistent results between v3 and v4 setups.

Prompt: - Create fixtures that run the same CSS: - Once with v3 tailwind.config.\* (no tokens in CSS). - Once with CSS tokens (simulated v4). - Validate: - version detection (auto chooses v4 when tokens found). - Coverage and classes are equivalent or produce approximate within thresholds. - Warnings include token-miss and v3-fallback where applicable. - Use SPEC.md “Tailwind v4 Migration Plan → Incremental migration steps” for guidance.

Acceptance criteria: - Test suite passes; parity demonstrated. - Differences are only where tokens or thresholds imply approximations.

#### 10) Developer Docs: SPEC.md Anchors and Comments

Goal: Keep SPEC.md the single source of truth during development.

Prompt: - Add internal comments in key modules (tokens loader, resolvers, ordering, variants) that reference SPEC.md sections by heading. - For complex mappings or edge cases, include a short comment referencing the relevant SPEC.md subsection.

Acceptance criteria: - New and updated files include SPEC.md references. - Future contributors can navigate from code to SPEC.md easily.

If you want, I can tailor these prompts to your current code layout (file paths, function names) by scanning your repo next.