### Pacote de prompts para Cursor IDE — “CSS → Tailwind v3 Core”

Abaixo estão prompts em etapas para você colar no Cursor IDE e ir iterando o projeto. A proposta cria um “core” desacoplado (TypeScript) com parseadores, normalizadores, matchers, rules engine, formatter e reporter — pronto para depois plugar em CLI e/ou UI.

Cada prompt indica: objetivo, instruções, critérios de aceite e arquivos esperados. Cole um prompt, deixe o Cursor gerar/editar os arquivos, rode testes localmente, depois avance para o próximo.

#### Prompt 1 — Inicializar monorepo mínimo (core + Playground)

Objetivo: Criar estrutura de projeto Node/TypeScript com pacote “core” isolado e “Playground” para dev local.

Prompt: ~~Crie um monorepo mínimo com pnpm workspaces contendo: - packages/core: biblioteca TypeScript “tailwindify-core” - apps/Playground: app Node simples para testar o core via linha de comando de desenvolvimento (não é o CLI final) Configurar: - TypeScript estrito, tsup para build do core, vitest para testes, eslint + prettier - pnpm workspaces, root package.json, tsconfig base - Editorconfig - Prettier com plugin “prettier-plugin-tailwindcss” instalado e configurado no root - Node 18+~~

~~Arquivos esperados: - package.json (root) com “workspaces” apontando para packages/\* e apps/\* - pnpm-lock.yaml (não precisa gerar conteúdo agora, só scripts) - tsconfig.base.json no root - .editorconfig, .gitignore, .prettierrc, .eslintrc.cjs - packages/core/package.json, tsconfig.json, src/index.ts - apps/Playground/package.json, tsconfig.json, src/index.ts Scripts: - Root: “build” (constrói core), “test” (vitest), “lint”, “format” - core: build com tsup (CJS+ESM), test com vitest - Playground: script “dev” que importa o core e executa uma demo simples~~

~~Critérios de aceite: - pnpm -w build compila o core - pnpm -w test roda vitest vazio - apps/Playground “dev” roda e imprime “Playground ready”~~

#### Prompt 2 — Definir contratos e arquitetura do core

Objetivo: Criar os tipos e interfaces base, incluindo Theme, MatchCtx, RuleHandler, resultado com avisos.

Prompt: ~~No pacote core, defina: - Tipos: - CssDeclaration { prop: string; value: string } - CssRule { selector: string; declarations: CssDeclaration[] } - TransformResult { classes: string[]; warnings: string[]; coverage: { matched: number; total: number } } - MatchCtx { theme: any; opts: { strict: boolean; approximate: boolean } } - RuleHandler: (value: string, ctx: MatchCtx) => string[] | null - Exporte funções: - toTailwind(prop: string, value: string, ctx: MatchCtx): string[] // com fallback [prop:value] - transformRule(rule: CssRule, ctx: MatchCtx): TransformResult - transformDeclarations(decls: CssDeclaration[], ctx: MatchCtx): TransformResult - Crie pastas core: src/core/{rulesEngine.ts, reporter.ts} - reporter.ts: util que calcula coverage e agrega warnings~~

~~Critérios de aceite: - Compila - Há testes unitários vazios preparados para essas funções~~

#### Prompt 3 — Normalizadores de valor e utilitários base

Objetivo: Criar normalização de valores e helpers para arbitrários.

Prompt: ~~Adicione em src/core/normalizers.ts: - normalizeValue(v: string): string — trim, lower-case quando aplicável, preserva unidades - isPx/isRem/isEm/isPct/isNumber - toArbitrary(prefix: string, v: string): string → ${prefix}-[${v}] - arbitraryProperty(prop: string, v: string): string → [${prop}:${v}] - parseBoxShorthand(v: string): string[] // split por espaço (1-4 valores) - parseColorNormalize(v: string): string // padroniza hex (#fff -> #ffffff), remove espaços~~

~~Crie testes vitest para cada função com casos comuns e edge cases.~~

~~Critérios de aceite: - vitest ok - funções cobrem casos básicos e não quebram inputs estranhos~~

#### Prompt 4 — Loader de tema Tailwind e acesso a tokens

Objetivo: Carregar tailwind.config.js opcionalmente e expor tokens para matching.

Prompt: ~~Crie src/core/themeLoader.ts com: - loadTheme(cwd: string): any - Tenta carregar tailwind.config.(js|cjs|mjs|ts) do cwd - Se não existir, retorna tema default mínimo: spacing padrão do Tailwind, screens sm/md/lg/xl/2xl, cores básicas (black, white, transparent, current) - getToken resolvers: - resolveSpacingToken(value: string, theme): string | null - resolveColorToken(hexOrNamed: string, theme): { type: ‘exact’|‘none’; class?: string } - resolveFontSizeToken(pxOrRem: string, theme): string | null - resolveLineHeightToken(value: string, theme): string | null Obs: Para cores, inicialmente só “named” básicos; se não achar, fallback arbitrário.~~

~~Crie testes simulando temas simples.~~

~~Critérios de aceite: - Carrega tema quando existir - Fallback funciona sem tema~~

#### Prompt 5 — Matchers: spacing, typography, display, position/inset

Objetivo: Implementar mapeamentos principais.

Prompt: ~~Implemente em src/core/matchers: - spacing.ts - matchSpacing(prefix: ‘m’|‘p’|‘w’|‘h’|‘top’|‘right’|‘bottom’|‘left’, raw: string, ctx: MatchCtx): string[] - Suporta shorthand (m/p) 1-4 valores → my/mx/mt/mr/mb/ml etc. - Tenta casar com theme.spacing (resolveSpacingToken); se não, usa arbitrário - typography.ts - matchTypography(kind: ‘font-size’|‘line-height’|‘letter-spacing’, value, ctx): string - Para font-size: tenta token, senão text-[...] - line-height: tenta token, senão leading-[...] - letter-spacing: tracking-\* ou tracking-[...] - layout.ts - display map: block, inline, inline-block, flex, grid, none - position map: static, relative, absolute, fixed, sticky - inset: top/right/bottom/left via spacing matcher~~

~~Inclua testes: entradas em px, rem, %, números, e shorthands.~~

~~Critérios de aceite: - Casos básicos passam - Saída consistente~~

#### Prompt 6 — Matchers: cores, background, border, radius

Objetivo: Cobrir cor de texto/bg, border color/width, border radius.

Prompt: ~~Implemente: - colors.ts - matchColor(prefix: ‘text’|‘bg’|‘border’, value, ctx): string - Suporta named básicos (black, white, transparent, current) - Hex/RGB/var(…) → se não houver mapeamento no tema, usar arbitrário: ${prefix}-[${value}] - borders.ts - width: border, border-0, border-2, senão border-[...] - color: reaproveitar matchColor(‘border’, v) - radius: rounded-none, rounded, rounded-full (com heurística 50%/9999px), senão rounded-[...] - backgrounds.ts - background-color → bg-\* - background-size: cover/contain/auto → bg-cover/bg-contain/bg-auto ou bg-[...] - background-position: center/top/right/bottom/left/combos → utilitários conhecidos; senão bg-[...] - background-image: url(…) → bg-[url(...)]~~

~~Testes incluidos.~~

~~Critérios de aceite: - Mapeamentos cobrem casos típicos - Fallback arbitrário correto~~

#### Prompt 7 — Flex/Grid e gaps

Objetivo: Cobrir layout modernos.

~~Prompt: Adicione em src/core/matchers/flexGrid.ts: - flex-direction: row/row-reverse/col/col-reverse → flex-row etc. - justify-content: start/end/center/space-between/space-around/space-evenly → justify-\* - align-items: start/end/center/baseline/stretch → items-\* - gap: usa spacing matcher para gap, gap-x, gap-y - grid-template-columns: números inteiros pequenos → grid-cols-N; valores não mapeáveis → grid-cols-[...] - place-content/place-items/place-self com mapeamentos básicos Testes com múltiplos cenários.~~

~~Critérios de aceite: - gap: 12px vira gap-3 se token existir, senão gap-[12px] - grid-template-columns: repeat(3, minmax(0, 1fr)) → grid-cols-3~~

#### Prompt 8 — Rules Engine integrando todos os matchers

Objetivo: Registrar handlers e orquestrar fallback.

Prompt: ~~Atualize src/core/rulesEngine.ts: - Registro de handlers por propriedade CSS: - display, position, top/right/bottom/left - margin/padding (shorthands e lados) - width/height/min/max width/height - color, background-color, border-color, border-width, border-radius - font-size, line-height, letter-spacing, font-weight (mapear 100-900 para utilitários quando possível; caso contrário font-[...]) - text-align → text-left/center/right/justify - object-fit → object-contain/cover/fill/none/scale-down - justify-content, align-items, gap, grid-template-columns - Qualquer propriedade sem handler → arbitraryProperty(prop, value)~~

~~Atualize transformDeclarations e transformRule para: - Deduplicar classes - Ordenar com uma ordem de grupos simples (typography, layout, spacing, border, background, effects) antes de passar por prettier (apenas em UI/CLI futuros; aqui use ordenação estável simples) - Calcular warnings: sempre que usar valor arbitrário ou propriedade sem handler, adicione um warning com detalhe~~

~~Testes cobrindo transformRule em blocos com várias declarações.~~

~~Critérios de aceite: - Resultado previsível e estável - Warnings informativos~~

#### Prompt 9 — Variantes responsivas e pseudo-classes (parcial)

Objetivo: Preparar o core para variantes sem acoplar a parse de seletores complexos.

Prompt: ~~No core, adicione util “withVariant(prefix: string, classes: string[]): string[]” que aplica sm:, md:, etc. ou hover:, focus: nos itens. - Exponha API que aceite metadata opcional por declaração: { prop, value, variants?: string[] } - Ajuste transformDeclarations para aplicar variants quando fornecidas - Não implemente parse de media queries ainda; só a mecânica~~

~~Testes: - Dado decls com variants [‘hover’], verifique hover:text-red-500 - Dado variants responsivas [‘sm’, ‘md’], aplica sm:... md:...~~

~~Critérios de aceite: - Variants funcionam quando fornecidas~~

#### Prompt 10 — Parser simples de CSS inline e de regras CSS

Objetivo: Prover parsers mínimos para viabilizar testes locais no Playground.

Prompt: ~~Implemente em src/parsers: - inlineCss.ts: parseInlineCss(style: string): CssDeclaration[] - split por ‘;’, parse ‘prop: value’ - ignora vazios e mantêm ordem - cssRules.ts: parseCssRules(css: string): CssRule[] - Use PostCSS para parsear CSS em regras simples (ignorar @media por agora) - Extraia selector + declarations~~

~~Adapte apps/Playground/src/index.ts para: - Ler da stdin um CSS inline ou um bloco de regra (detecte presença de “{”) - Usar o core para transformar e imprimir: - Classes resultantes - Warnings - Cobertura~~

~~Critérios de aceite: - Rodar “pnpm -w –filter Playground dev” e testar entradas~~

#### Prompt 11 — Casos complexos e shorthands adicionais

Objetivo: Expandir cobertura.

Prompt: ~~Adicione suporte a: - border shorthand: “border: 1px solid #000” → border, border-[1px] ou border se token 1px for “border” default, cor via border-color - background shorthand: parse “background: url(x) center/cover no-repeat #fff” e mapear partes suportadas, resto arbitrário - overflow: visible/hidden/scroll/auto → overflow-\* (x/y específicos) - z-index: tokens comuns (z-0, z-10, z-50) senão z-[...] - opacity: opacity-0..100 quando casar, senão opacity-[...]~~

~~Inclua testes.~~

~~Critérios de aceite: - Shorthands quebrados de forma previsível - Fallback correto~~

#### Prompt 12 — Approximate vs strict matching

Objetivo: Implementar opção de aproximar para tokens do tema.

Prompt: ~~No themeLoader/matchers: - Adicionar função findNearestToken(valuePxOrRem, tokenMap) que escolhe token mais próximo e retorna também a diferença - Em MatchCtx.opts.approximate = true: - Para spacing e font-size/line-height: se diferença <= 1px (configurável), usar token e registrar warning “approximate mapping: 17px → text-sm (16px)” - Em strict = true: nunca aproximar; usar arbitrário~~

~~Testes com casos 15px→4 (1rem=16px), etc.~~

~~Critérios de aceite: - Com approximate on, mapeia para tokens próximos + warning - Com strict, não aproxima~~

#### Prompt 13 — Reporter e métricas

Objetivo: Medir qualidade da conversão.

Prompt: ~~Melhore reporter.ts: - coverage: matched = número de declarações que resultaram em classe não-arbitrária; total = total de declarações - stats por tipo: spacing, color, typography, layout, border, background, effects - warnings categorizados: arbitrary-value, no-handler, approximate - Expor summarize(result: TransformResult | TransformResult[]): string — retorna string legível com métricas~~

~~Teste unitário com um conjunto de declarações.~~

~~Critérios de aceite: - summarize retorna dados úteis - métricas batem com esperado~~

#### Prompt 14 — Documentação inicial

Objetivo: Criar README do core e instruções de uso do Playground.

Prompt: ~~Crie README.md no root e em packages/core contendo: - Objetivo do projeto - Arquitetura resumida - API pública do core - Como rodar Playground - Como estender handlers (exemplos) - Roadmap (CLI e UI futuros)~~

~~Critérios de aceite: - Docs claras e concisas~~

#### Prompt 15 — Pacote pronto para consumo (build e types)

Objetivo: Garantir artefatos corretos.

Prompt: ~~Ajuste tsup para gerar: - ESM e CJS outputs - DTS (declarações de tipos) - package.json do core com “exports” para ESM/CJS, “types” apontando para d.ts - Tree-shaking e sourcemaps~~

~~Valide que: - import { transformDeclarations } from ‘tailwindify-core’ funciona no Playground - Tipos são reconhecidos no editor~~

~~Critérios de aceite: - Build gera dist com .js e .d.ts - Imports funcionam em ESM e CJS~~

#### Prompt 16 — Testes de snapshot com fixtures reais

Objetivo: Consolidar qualidade.

Prompt: ~~Crie pasta tests/fixtures com blocos de CSS representativos (botão, card, grid). - Escreva testes que: - parseiam as regras com parseCssRules - aplicam transformRule - comparam classes resultantes via snapshots - Adicione variação strict vs approximate~~

~~Critérios de aceite: - Snapshots estáveis - Fácil detectar regressões~~

#### Prompt 17 — Plano para CLI/UI futuros (esqueleto sem implementar)

Objetivo: Deixar ganchos e interfaces prontas para expansão.

Prompt: - ~~No core, exporte também uma função high-level: - transformCssText(css: string, ctx: MatchCtx): { bySelector: Record<string, TransformResult> } - Crie pacote vazio “packages/cli” com README dizendo que usará o core - Crie “apps/web” vazio com README dizendo que usará o core em um worker (Monaco/React) - Não implementar agora; apenas placeholders e scripts básicos de build lint~~

~~Critérios de aceite: - Compila - Docs dos próximos passos claras~~

#### Prompt 18 — QA final e scripts DX

Objetivo: Produtividade no dia a dia.

Prompt: - ~~Adicione scripts no root: - “check”: lint + typecheck + test - “ci”: build + check - Configure Husky + lint-staged para format/lint em commits - Adicione exemplo de comando do Playground: - pnpm -w –filter Playground dev — entrada: - inline: color: #1f2937; font-size: 18px; line-height: 1.4; margin: 10px 20px; - regra: .btn { display: flex; justify-content: center; gap: 12px; } - Saída: classes e warnings~~

~~Critérios de aceite: - Fluxo local redondo - Commits com hooks funcionando~~

### Dicas para usar no Cursor IDE

* Cole um prompt por vez e deixe o Cursor criar/editar arquivos.
* Sempre peça para gerar também os testes quando criar um módulo novo.
* Após cada etapa, rode localmente: pnpm i, pnpm -w build, pnpm -w test, pnpm -w –filter Playground dev.
* Se o Cursor gerar caminhos errados, peça correção e consistência com a estrutura.

Se quiser, te preparo agora um repositório inicial em texto (arquivos e conteúdos) para você só copiar/colar. Ou posso adaptar os matchers iniciais para cobrir exatamente o seu design system (spacing/cores/fontes) desde já.