

13 Güçlü Claude Code Promptu

PRD yazmaktan debug etmeye, E2E testlerinden tekrar eden işleri skill'e dönüştürmeye kadar kullanabileceğin kopyalanabilir promptlar.

Ön Koşullar

Bu promptların çoğu, proje klasörünü doğrudan okuyabilen ve terminal komutları çalıştırabilen Claude Code içinde kullanılmak üzere hazırlanmıştır.

- Claude Code erişimi: Claude Code kurulu olmalı ve uygun bir Claude/Anthropic hesabıyla oturum açmalısın. Erişim ve kullanım şekli hesabına göre değişebilir.
- Proje klasörü: Claude Code'u çalıştırmadan önce doğru repo veya proje klasöründe olduğundan emin ol. Git kullanman geri dönüşleri kolaylaştırır.
- Köşeli parantezler: **[BU ŞEKİLDE]** yazılan alanları kendi projenin bilgileriyle değiştir.
- Dosya ve komut izinleri: Bazı promptlar dosya değiştirebilir, migration çalıştırabilir, test koşturabilir veya servis bağlayabilir. Planı ve diff'i incelemeden kritik değişiklikleri onaylama.
- Güvenlik: Güvenlik denetimi promptunu yalnızca inceleme yetkin olan kod tabanlarında kullan. API anahtarlarını promptta yazma; environment variable kullan.

Özellikler ve hesap erişimi zamanla değişebilir; kendi Claude Code sürümünde görünen seçenekleri esas al.

01

PROMPT 01

PRD Yazdır

Yeni bir özellik için problem, kapsam, kullanıcı hikâyeleri ve başarı ölçütlerini tek dosyada netleştirir.

Write a complete PRD for the feature below.

Feature: **[DESCRIBE FEATURE]**

Users: **[WHO USES IT]**

Stack: **[YOUR STACK]**

Include:

- * Problem statement + success metrics
- * User stories with acceptance criteria
- * Scope: what ships in v1, what does not
- * Data model changes
- * Edge cases + failure states
- * Open questions for me to answer

Keep it under 2 pages. Be specific, no filler.

Save it as docs/prd-**[feature]**.md so every later prompt can reference it.

02

PROMPT 02

Claude.md Dosyası Oluştur

Claude Code'un projeyi, komutları, mimariyi ve ekip kurallarını her oturumda daha doğru anlaması için CLAUDE.md üretir.

Scan this entire codebase, then generate a CLAUDE.md.

Include:

- * What this project is, in 2 lines
- * Tech stack + the versions that matter
- * Commands: **[DEV / BUILD / TEST / LINT]**
- * Architecture: where things live and why
- * Code conventions you actually detect in the code
- * Hard rules: **[YOUR NON-NEGOTIABLES]** -- what to never touch without asking
- * Gotchas a new engineer would hit in week 1

Write rules as short imperatives. Nothing generic -- only what is true for THIS repo. If you are unsure about a rule, ask me instead of inventing it.

03

PROMPT 03

Ultra Plan Modu

Kod yazmadan önce etkilenecek dosyaları, alternatif yaklaşımları, riskleri ve geri dönüş planını çıkartır.

Enter plan mode. Do NOT write any code yet.

Task: **[PASTE TASK]**

Constraints: **[DEADLINE / STACK / NO-GO ZONES]**

1. Read every file this task touches, list them with one line on what each does today
2. Map current behavior vs target behavior
3. Propose 2-3 approaches with real tradeoffs: complexity, risk, blast radius
4. Pick one and justify it in 3 lines
5. Break it into steps small enough to verify one at a time, each with its own check
6. List risks + the exact rollback for each
7. Flag anything touching **[AUTH / PAYMENTS / PROD DATA]** for my explicit sign-off

Then stop. Show me the plan and wait for my approval before touching a single file.

04

PROMPT 04

UI/UX Tasarım Briefi

Bir ekran veya akış için kullanıcı yolculuğundan tasarım tokenlarına kadar uygulanabilir bir UI/UX briefi oluşturur.

Create a full UI/UX design brief for: **[SCREEN OR FLOW]**

Audience: **[USERS]**

Brand: **[COLORS / FONTS / VIBE]**

Deliver:

- * User journey through the flow, step by step
- * Layout per screen: hierarchy, spacing, breakpoints
- * Component inventory with every state -- hover, empty, error, loading
- * Typography + color tokens
- * Motion: what animates, duration, easing
- * Accessibility notes

Study patterns from **[2-3 PRODUCTS YOU ADMIRE]** for direction. Never copy them.

05

PROMPT 05

Kurulum Planı

Onaylanmış PRD veya teknik dokümanı küçük, doğrulanabilir geliştirme adımlarına böler.

Create an implementation plan for: **[APPROVED SPEC / PRD]**

Rules:

- * Sequence steps so the app compiles and runs after EVERY single step
- * Each step: files touched, what changes, how I verify it works
- * Flag steps needing a migration or new dependency
- * Put the riskiest unknowns first
- * Size each step: **[S / M / L]**

Output a numbered build sequence I can run one step at a time. Wait for my "go" between steps.

06

PROMPT 06

MCP Server Baęla

Bir servis veya API için uygun MCP sunucusunu bulur; yoksa gerekli araçları ve güvenli yapılandırmayı kuracak planı çıkarır.

Wire up an MCP server for: **[SERVICE / API]**

What I need it to do: **[JOBS TO BE DONE]**

1. Check for an official or well-maintained existing server first -- name your source
2. If one exists: exact install command + the .mcp.json config, scoped to this project
3. If not: scaffold one with the MCP SDK -- tools, auth, error handling, typed responses
4. Add ONLY the tools I will actually use: **[LIST]**
5. Wire secrets through **[ENV VARS]**, never hardcode keys in the config file
6. Verify the connection and call one tool end to end, show me the output
7. Document each tool in 1 line so future sessions know when to reach for it

Veri Tabanı Bağla

Uygulamaya veri tabanı bağlantısı, şema, migration, erişim kuralları ve doğrulama akışı ekletir.

Connect this app to **[POSTGRES / SUPABASE / YOUR DB]**.

- * Pick the client that fits this stack, justify it in 1 line
- * Env vars: name them, add to .env.example, never commit real values
- * Schema: tables for **[ENTITIES]** with types, relations, and indexes for **[HOT QUERIES]**
- * Create + run the migrations, and show me the rollback for each one
- * One typed query helper per table -- no raw SQL scattered through components
- * Access rules / row-level security if this is **[MULTI-TENANT]**
- * Connection pooling if we are serverless

Then prove it: seed one row, read it back, show me the output.

08

PROMPT 08

Güvenlik Açıklarını Bul

Yetkili olduğun bir kod tabanında kimlik doğrulama, veri sızıntısı, injection ve bağımlılık açıklarını sistemli biçimde taratır.

Audit this codebase for security gaps.
Attack it like you want in.

Focus areas: **[AUTH / PAYMENTS / USER DATA]**

Check:

- * Secrets in code, config, or git history
- * Injection: SQL, XSS, command, path traversal
- * Auth: routes missing checks, weak sessions, broken redirects
- * IDOR: can user A read user B's data?
- * File uploads + input validation on every form
- * Dependency CVEs -- run the audit, read it
- * Rate limiting on **[EXPENSIVE ENDPOINTS]**
- * What leaks through error messages and logs

Rank findings by severity with exact file:line, fix the critical ones now, and list the rest as tickets with effort estimates.

Debug Et

Tahmin yürütmek yerine stack trace, hipotez, log ve küçük testlerle hatanın kök nedenine ulaşır.

Debug this error. Do NOT guess.

Error: **[PASTE FULL ERROR + STACK TRACE]**

When it happens: **[STEPS TO REPRODUCE]**

1. Read the stack trace, open the exact files involved
2. State expected vs actual behavior in 1 line
3. List 3 hypotheses, ranked by likelihood
4. Prove or kill each one with logs or a tiny test -- evidence, not vibes
5. Fix the root cause, not the symptom
6. Search the repo for the same pattern -- if it can break here, it breaks elsewhere
7. Add a regression test that fails without the fix
8. Tell me in 2 lines why it broke and why it can never break this way again

10

PROMPT 10

Uygulamayı Test Et

Playwright ile ana kullanıcı akışlarını, hata senaryolarını ve CI davranışını kapsayan E2E testleri yazdırır.

Write Playwright E2E tests for: **[AKIŞ]**

Stack: **[KULLANDIĞIN STACK]** CI: **[GITHUB ACTIONS / DİĞERİ]**

- * Money paths first: **[KAYIT OLMA / ÖDEME / ANA AKSİYON]**
- * Test what the user sees, not implementation details
- * Selectors: roles and labels, never brittle CSS chains
- * One unhappy path per flow: bad input, network failure, expired session
- * Tests stay independent -- any order, zero shared state, each seeds its own data
- * Headless in CI, headed locally for debugging
- * Screenshots + traces on failure only

Run the suite, show me the results, fix what fails, and tell me what the suite still does NOT cover.

11

PROMPT 11

Ölü Kodu Temizle

Kullanılmayan kodu, bağımlılıkları, CSS sınıflarını ve eski feature flag'leri güvenli şekilde temizletir.

Find and delete dead code in this repo.

Scope: **[TÜM REPO / BELİRLİ KLASÖR]**

- * Unused exports, components, hooks, utils
- * Unreachable branches + commented-out blocks
- * package.json dependencies nothing imports
- * Stale feature flags stuck always-on or always-off
- * Duplicate logic that should merge into one
- * Dead CSS classes and unused assets

Verify with a search before EVERY deletion -- dynamic imports and string references count.

Delete in small commits, run **[BUILD + TESTLER]** after each one, and report total lines removed plus anything you were not 100% sure about.

12

PROMPT 12

Temiz Git Commitleri Yaz

Staged değişiklikleri mantıksal parçalara ayırır ve seçtiğin commit standardına uygun mesajlar üretir.

Commit my staged changes properly.

Convention: **[CONVENTIONAL COMMITS / KENDİ FORMATIN]**

- * Split unrelated changes into separate commits
- * Format: type(scope): what changed and why
feat / fix / refactor / chore / docs / test
- * Subject under 50 chars, imperative mood
- * Body explains the WHY, wrapped at 72
- * Reference the ticket: **[TICKET NUMARASI]**
- * Never mix a refactor with a behavior change in one commit
- * Never commit **[GİZLİ BİLGİLER / .ENV / OLUŞTURULAN DOSYALAR]**

Show me the plan -- files per commit + messages before you commit anything.

Then commit one at a time so I can stop you between them.

13

PROMPT 13

Görevi Bir Skill'e Dönüştür

Sık tekrarladığın bir işi proje seviyesinde Claude Code skill'ine dönüştürür ve gerçek örnek üzerinde denetir.

Tekrarladığım bu görevi bir Claude Code skill'ine dönüştür.

Sürekli yaptığım görev: **[GÖREVİ VE ADIMLARI AÇIKLA]**

- * `.claude/skills/[İSİM]/SKILL.md` oluştur
- * Frontmatter: gerçekten kullandığım tetikleyici ifadeleri içeren isim + açıklama
- * Body: numaralı iş akışı, kullandığım kurallar, edge case'ler
- * Benden neyi sorması gerektiği ve neyi kendi başına çıkarabileceği
- * 'Done' durumunun nasıl görüldüğü

Sonra skill'i gerçek bir örnek üzerinde dry-run yap ve çıktı, bunu manuel yaptığım şekilde eşleşene kadar iyileştir.