

Beğendiğin uygulamanın içini REA ile aç

REA (Reverse Engineer Anything), kaynak kodunu görmeden bir uygulamanın nasıl çalıştığını çıkaran açık kaynak bir araç. Claude Code'a kurulum, ilk soru, terminalden hızlı tarama, Wispr Flow örneği ve bulguyu kendi projene taşıma.

SÜRE

25 dk

SEVİYE

Orta

ADIM

5

PROMPT

3

Bir uygulamada bir özellik görüyorsun ve "bunu nasıl yapmışlar?" diye merak ediyorsun. Kaynak kodu elinde yok. Bu rehberde REA'yı Claude Code'a bağlayıp bir uygulamanın içini nasıl açacağını anlattım. Ben bunu Wispr Flow üzerinde denedim; bulduklarımı ve REA'nın nerede yetmediğini aşağıda dürüstçe yazdım. Komutların ve prompt'ların sağ üstündeki **Kopyala** butonuna bas, yapıştır, devam et.

Ne yapacağız

Bir masaüstü uygulamasını seçip "şu özellik nasıl çalışıyor?" sorusunu cevaplayacağız. Sonra öğrendiğini kendi projende kendi kodunla kuracağız.

Tersine mühendislik ne? Bitmiş bir ürünü parçalarına ayırıp nasıl yapıldığını çözmek. Yazılımda bu, kaynak kodu olmadan uygulamanın dosyalarına bakıp hangi parçanın neyi yaptığını bulmak demek.

Her uygulama aslında bir klasör. Mac'te bir uygulamaya sağ tıklayıp **Paket İçeriğini Göster** dersin içindeki dosyaları görürsün. Slack, Obsidian, VS Code, Claude ve Wispr Flow gibi Electron ile yazılmış uygulamaların kodu bu klasörde JavaScript olarak duruyor. Okunması zor (sıkıştırılmış) ama okunabilir.

REA ne yapıyor? [REA](#) açık kaynak (MIT lisanslı) bir araç. Ajanına MCP üzerinden bağlanıyor ve şunları inceleyebiliyor:

- Electron ve JavaScript uygulamaları: modüller, import'lar, uygulamanın kendi içindeki mesajlaşması (IPC)
- Web siteleri: sayfa yapısı, script'ler, ağ istekleri
- Derlenmiş programlar: Hopper, Ghidra ya da IDA kuruluysa onlar üzerinden
- .NET, Android APK, SQLite veritabanı ve başka birkaç format

Analiz senin bilgisayarında çalışıyor. REA her bulgunun yanında kanıtını (dosya, satır, bağlantı) ve neyi bilemediğini de veriyor.

Sonuçta elinde üç şey olacak:

1. Seçtiğin özelliğin adım adım nasıl çalıştığı
2. Her adımın kanıtı (hangi dosyada, hangi çağrıyla)
3. Aynı fikri kendi projende kendi kodunla kurmak için bir plan

MALZEMELER

- [Claude Code](#). Codex, Cursor ve Gemini CLI de destekleniyor.
- Node.js 22.19 ya da üstü ve npm. Kontrol için: `node -v`

- İncelemek istediğin uygulama. Bu rehberdeki örnek Mac'te; Windows ve Linux'ta da çalışıyor.

ARAÇLAR

Claude Code

REA

Node.js

01 REA'yı kur

Terminalde:

TERMINAL

```
npx rea-agents setup
```

Kurulum sana hangi ajanları bağlamak istediğini soruyor. **Claude Code**'u seç. Değişiklik yapmadan önce hangi dosyaya ne yazacağını gösterip onayını istiyor, mevcut ayarlarının yedeğini de alıyor. Hopper kurmayı da teklif ediyor; bu isteğe bağlı. Yalnız JavaScript/Electron uygulamalarına bakacaksan gerek yok.

Kurulum bitince **Claude Code'u kapatıp yeniden aç.**

Kurulumu Claude'a da yaptırabilirsin:

PROMPT

morluto/rea reposunu (REA: Reverse Engineer Anything) Claude Code'a kur. Resmi kurulum komutu ``npx rea-agents setup``. Kurulum hangi dosyaları değiştirecekse önce bana göster, onay vermeden yazma. Bitince yeniden başlatmam gerekiyorsa söyle.

02 İlk soruyu sor

Claude Code'u yeniden açtıktan sonra şu kalıbı kullan. Uygulama adını ve merak ettiğin özelliği değiştir:

PROMPT

REA ile Wispr Flow uygulamasını incele (/Applications/Wispr Flow.app). Konuşmayı bitirdiğinde yazının imlecin olduğu kutuya nasıl yazıldığını bul.

1. Bu akışı adım adım çıkar: hangi parça hangisini çağırıyor.
2. Her adım için kanıt göster: dosya yolu ve ilgili fonksiyon ya da mesaj adı. Uzun kod parçası kopyalama, 2-3 satır yeter.
3. Emin olmadığın ya da REA'nın göremediği yerleri ayrıca yaz.
4. Uygulamanın içinde ayrı çalışan bir program varsa onu da belirt.

Soru ne kadar dar olursa cevap o kadar iyi. "Bu uygulama nasıl çalışıyor?" yerine "şu butona basınca ne oluyor?" diye sor.

03 Terminalden hızlı tarama (isteğe bağlı)

Claude'a sormadan önce uygulamanın genel resmini görmek istersen REA'yı doğrudan terminalden çalıştırabilirsin. Electron uygulamalarında kod `app.asar` dosyasında duruyor:

TERMiNAL

```
npx -y rea-agents@latest analyze-javascript-application  
"/Applications/Wispr Flow.app/Contents/Resources/app.asar" --  
json > wispr.json
```

Çıktı büyük (Wispr Flow'da 59 MB). Özetini görmek için:

TERMiNAL

```
jq -c '{source: {kind: "inline", evidence: .}, view: {kind:  
"summary"}}' wispr.json > ozet-istek.json  
npx -y rea-agents@latest inspect-analysis-view ozet-istek.json
```

Bende tarama yaklaşık 82 saniye sürdü ve şunu verdi: 167 JavaScript dosyası, 1.428 modül, 207 iç çağrı (IPC).

04 Bende ne çıktı: Wispr Flow nasıl yazıyor?

Wispr Flow'da bir kısayol tuşuna basıp konuşuyorsun, bıraktığında yazı hangi uygulamadaysan oraya yazılıyor. İçini açınca şu akış çıktı:

1. **Konuşursun.** Ses yazıya çevriliyor.
2. **Yapay zekâ metni düzeltir.** Bu adım başarısız olursa ham metin kullanılıyor (kodda `PrePastePolish`).

3. **İmleç yazılabilir bir kutuda mı?** Uygulama önce bunu soruyor (`getTextBoxInfo`). Değilse metni yapıştırmıyor, bildirim olarak gösteriyor.
4. **Yazıyı gizli küçük bir program yazıyor.** Metni Electron'un kendisi yapıştırmıyor. Paketin içinde `swift-helper-app-dist` klasöründe Swift ile yazılmış ayrı bir uygulama var; ona `PasteText` isteği gidiyor. O program kapanmışsa yeniden başlatılıyor (`pasteTextAndRelaunchHelperIfNeeded`).
5. **Gerçekten yazıldı mı?** Yapıştırmadan önce ve sonra kutudaki metne bakılıp yazının gerçekten çıktığı kontrol ediliyor.

Bunu kendin de görebilirsin: Finder'da Uygulamalar → Wispr Flow'a sağ tık → **Paket İçeriğini Göster** → Contents → Resources → `swift-helper-app-dist`. İçinde ikinci bir "Wispr Flow" uygulaması duruyor.

REA'nın yetmediği yer. REA'nın "özellik izle" komutu (`trace-application-feature`) bu akışı bulamadı; denediğim 6 başlangıç noktasının hepsi 0 sonuç döndü. REA da çıktısında bu uygulama için çıkardığı haritanın eksik olduğunu söylüyordu. Yukarıdaki zinciri, REA'nın taramasından sonra paketlenmiş kodun içinde fonksiyon ve mesaj adlarını arayarak bulduk. Yani REA tek başına cevabı vermiyor; ajanla birlikte kullanınca işe yarıyor. Ajanın verdiği her adımı kanıtlı kontrol et.

05 Öğrendiğini kendi projene taşı

Akışı anladıktan sonra aynı fikri kendi kodunla kur:

PROMPT

Wispr Flow'da bulduğumuz akışı (metni hazırla → hedef kutu yazılabilir mi kontrol et → yazmayı ayrı küçük bir yardımcı programa yaptır → yazıldığını doğrula) kendi projem için tasarla.

1. Wispr Flow'un kodunu kopyalama; yalnız mimariyi örnek al.
2. Projeimin diline ve yapısına uygun en sade çözümü öner.
3. Hangi adımın işletim sistemi izni gerektirdiğini (ör. Mac'te Erişilebilirlik) ayrıca yaz.
4. Kodu yazmadan önce planı bana göster.

Bu kalıp yalnız konuşmayı yazıya çeviren uygulamalar için değil. Duolingo'daki bir alıştırmanın, Instagram'daki bir akışın ya da sevdiğin bir masaüstü araçındaki bir özelliğin mimarisini de aynı yolla çözebilirsin.

Önce: neyi inceleyebilirsin?

REA kendi bilgisayarındaki, kendi kurduğun uygulamayı okuyor. Yine de:

- Bunu **öğrenmek** için kullan. Bulduğun kodu kopyalayıp kendi ürününe koyma; fikri anla, kendi kodunla yaz.
- Uygulamanın kullanım koşullarına bak. Bazıları tersine mühendisliği açıkça yasaklıyor.
- Şifreleri, lisans kontrollerini ya da güvenlik önlemlerini aşmak için kullanma.

PÜF NOKTALARI

artifact_integrity_mismatch hatası. Uygulama pakettlendikten sonra içindeki bazı dosyalar imzalanmış ya da değişmiş olabiliyor; REA bunu fark

edip duruyor. Kendi kurduğun, güvendiğin bir uygulamaysa komuta `-- integrity-policy record-and-continue` ekle. REA farkı kaydedip devam ediyor.

Node sürümü hatası. REA Node.js 22.19 ya da üstünü istiyor. `node -v` ile kontrol et, eskiyse güncelle.

Uygulamanın kodu JavaScript değil. Safari, Notlar gibi uygulamalar makine koduna derlenmiş. Bunlar için REA'nın Hopper, Ghidra ya da IDA ile çalışması gerekiyor. Kurulum sırasında Hopper'ı kurmayı kabul edebilir ya da var olan Ghidra'yı gösterebilirsin.

"İzle" komutu 0 sonuç veriyor. Bu, özelliğin olmadığı anlamına gelmiyor; REA da bunu çıktısında söylüyor. Ajandan, REA'nın taramasından sonra paket içindeki fonksiyon ve mesaj adlarını aramasını iste.

Cevap çok genel. Soruyu daralt: tek bir buton, tek bir kısayol, tek bir ekran.

BİTTİ Mİ?

- ☐ Node.js 22.19+ kurulu
- ☐ `npx rea-agents setup` ile REA'yı Claude Code'a bağladım ve Claude Code'u yeniden başlattım
- ☐ İncelediğim uygulamanın kullanım koşullarına baktım
- ☐ Soruyu tek bir özelliğe daralttım
- ☐ Her adımın kanıtını (dosya, fonksiyon, mesaj adı) gördüm
- ☐ REA'nın göremediği yerleri ayrıca not ettim
- ☐ Fikri kendi projemde kendi kodumla kurdum, kodu kopyalamadım

Takıldığın yeri Instagram'da yaz, rehberine eklerim.