

# Codex Görev Otomasyonu (Node.js / TypeScript)

Aşağıdaki proje, anlattığınız döngüyü **tam otomatik** yürütmek için hazırlanmıştır:

- Bir **görev listesi** (Markdown) ve bir **repository** var.
- Orkestratör, listedeki sıradaki görevi **Codex**'e verir.
- Görev **başarıyla** bittiğinde listedeki bir sonraki göreve geçer.
- **Her 3 başarılı görev** sonunda “create pr” akışını tetikler (isteğe göre **Codex UI** üzerinden ya da doğrudan **GitHub API** ile).
- Liste **bittiğinde**, belirlediğiniz **prompt** ile Codex'e “ask” diyerek **yeni görev listesi** alır ve aynı döngüyü sürdürür.

**Not:** “Codex” için iki sürücü (driver) katmanı tanımlıdır. Birincisi, gerçek bir API varsa onu kullanmanız için **API tabanlı** bir şablon. İkincisi, web arayüzü olan bir ürün ise (ör. bir “Codex” UI) **Playwright** ile etkileşim sağlayan **UI Otomasyonu** şablonu. Hangisini kullanacağınıza `CODEX_DRIVER` ayarıyla karar verirsiniz.

## Hızlı Kurulum

1. **Depoyu hazırlayın** (bu orkestratörü ayrı bir repo veya monorepo içi paket olarak tutabilirsiniz).
2. Node 18+ kurulu olsun. Paket yöneticiniz `pnpm` / `npm` / `yarn` olabilir.
3. `.env` dosyasını doldurun (aşağıda örnek var).
4. `pnpm i` (veya `npm i`), ardından `pnpm start` (veya `npm run start`).
5. Çalışma sırasında **Codex** UI kullanacaksanız, ilk koşuda Playwright oturum açma adımlarını yorumlarda belirtildiği gibi uyarlayın.

## Proje Yapısı

```
.
├─ package.json
├─ tsconfig.json
├─ .env.example
├─ README.md
├─ src/
│   ├─ config.ts
│   ├─ state.ts
│   ├─ codex.ts
│   ├─ github.ts
│   └─ index.ts
├─ tasks/
│   └─ todo.md
```

```
└─ prompts/  
  └─ new_list_prompt.txt
```

## **.env.example**

```
# Zorunlu  
GITHUB_TOKEN=ghp_...  
GITHUB_OWNER=owner  
GITHUB_REPO=repo  
BASE_BRANCH=main  
  
# PR stratejisi: "codex" -> Codex UI üzerinden PR akışı, "github" -> doğrudan  
# GitHub API  
PR_STRATEGY=codex  
  
# Codex driver: "api" veya "ui"  
CODEX_DRIVER=ui  
  
# CODEX API varsa  
CODEX_API_BASE=https://codex.example.com/api  
CODEX_API_KEY=...  
  
# UI otomasyonu (Playwright) gerekiyorsa  
CODEX_UI_URL=https://codex.example.com/  
CODEX_UI_EMAIL=you@example.com  
CODEX_UI_PASSWORD=your-password  
# Alternatif: mevcut oturumu cookie ile taşımak isterseniz (opsiyonel)  
CODEX_UI_COOKIES_JSON=./cookies.json
```

## **package.json**

```
{  
  "name": "codex-orchestrator",  
  "version": "1.0.0",  
  "private": true,  
  "type": "module",  
  "scripts": {  
    "build": "tsc -p .",  
    "start": "tsx src/index.ts",  
    "start:prod": "node dist/index.js",  
    "lint": "eslint ."
```

```
},
"dependencies": {
  "@octokit/rest": "^20.1.1",
  "dotenv": "^16.4.5",
  "fast-glob": "^3.3.2",
  "marked": "^12.0.2",
  "zod": "^3.23.8"
},
"devDependencies": {
  "@types/node": "^20.12.12",
  "eslint": "^9.9.0",
  "playwright": "^1.47.2",
  "tsx": "^4.16.2",
  "typescript": "^5.5.4"
}
}
```

## tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "ESNext",
    "moduleResolution": "Bundler",
    "esModuleInterop": true,
    "forceConsistentCasingInFileNames": true,
    "strict": true,
    "skipLibCheck": true,
    "outDir": "dist",
    "resolveJsonModule": true
  },
  "include": ["src"]
}
```

## README.md

# Codex Görev Orkestratörü

Bu araç, `tasks/todo.md` içindeki görevleri sırayla **Codex**'e gönderir, her başarılı 3 görevde **PR** akışını tetikler, görevler bittiğinde `prompts/new_list_prompt.txt` içeriğini **ask** ederek yeni görev listesi ister ve

döngüyü sürdürür.

## Çalıştırma

```
```bash
cp .env.example .env
# .env içini doldurun
pnpm i
pnpm start
```

## Ayarlar

- PR\_STRATEGY=codex | github
- CODEX\_DRIVER=ui | api

## Görev Listesi Formatı

tasks/todo.md satırları şu formatta olmalı:

- [ ] Login sayfasına şifre görünür/gizli toggle ekle
- [ ] Ürün kartında fiyat biçimlendirmesi 9,99 → 9,99 ₺
- [ ] Sepet boşsa CTA butonunu devre dışı bırak

Tamamlananlar [x] ile işaretlenir.

---

## tasks/todo.md (örnek)

- ```
```md
- [ ] Kullanıcı profilinde avatar güncelleme hatasını düzelt
- [ ] Sipariş detayı sayfasında toplam KDV hesaplamasını düzelt
- [ ] Admin paneli log listesinde tarih sıralamasını DESC yap
- [ ] Fatura PDF çıktısında şirket ünvanını büyük harf yap
- [ ] Base modal bileşenine ESC ile kapatma ekle
- [ ] 404 sayfasında ana sayfaya dönüş linki ekle
```

## prompts/new\_list\_prompt.txt (örnek)

Bu repository için tamamlanmış işleri göz önüne alarak, **\*\*küçük ama bağımsız\*\*** 6-10 maddelik yeni bir geliştirme/iyileştirme listesi üret. Her madde tek

cümlelik, net bir çıktı hedeflesin ve tek commit ile yapılabilir ölçekte olsun. Yanıtı sadece maddeler halinde ver.

## src/config.ts

```
import 'dotenv/config';

export const CONFIG = {
  owner: process.env.GITHUB_OWNER!,
  repo: process.env.GITHUB_REPO!,
  baseBranch: process.env.BASE_BRANCH || 'main',
  prEvery: 3,
  tasksFile: 'tasks/todo.md',
  newListPromptFile: 'prompts/new_list_prompt.txt',
  prStrategy: (process.env.PR_STRATEGY || 'codex') as 'codex' | 'github',
  codex: {
    driver: (process.env.CODEX_DRIVER || 'ui') as 'ui' | 'api',
    apiBase: process.env.CODEX_API_BASE,
    apiKey: process.env.CODEX_API_KEY,
    uiUrl: process.env.CODEX_UI_URL,
    uiEmail: process.env.CODEX_UI_EMAIL,
    uiPassword: process.env.CODEX_UI_PASSWORD,
    cookiesJson: process.env.CODEX_UI_COOKIES_JSON,
  },
};

export function assertConfig() {
  const required = ['GITHUB_TOKEN', 'GITHUB_OWNER', 'GITHUB_REPO'];
  for (const k of required) if (!process.env[k]) throw new Error(`Missing env: ${k}`);
}
```

## src/state.ts

```
import { promises as fs } from 'fs';

export type OrchestratorState = {
  nextTaskIndex: number; // sıradaki TODO satır indeksi
  successInBatch: number; // bu batch içinde kaç başarı var
  batchNumber: number; // 1,2,3...
  totalCompleted: number;
```

```

};

const STATE_PATH = '.codex-state.json';

export async function loadState(): Promise<OrchestratorState> {
  try {
    const raw = await fs.readFile(STATE_PATH, 'utf8');
    return JSON.parse(raw) as OrchestratorState;
  } catch {
    const init: OrchestratorState = { nextTaskIndex: 0, successInBatch: 0,
batchNumber: 1, totalCompleted: 0 };
    await saveState(init);
    return init;
  }
}

export async function saveState(s: OrchestratorState) {
  await fs.writeFile(STATE_PATH, JSON.stringify(s, null, 2), 'utf8');
}

```

## src/codex.ts

```

import { chromium, Browser, Page } from 'playwright';
import { CONFIG } from './config.js';

export type Task = { title: string };

export interface CodexClient {
  submitTask(repoFullName: string, task: Task): Promise<{ jobId: string }>;
  waitForCompletion(jobId: string, opts?: { timeoutMs?: number }):
Promise<'success' | 'failed'>;
  createPR(opts: { title: string; body: string }): Promise<void>;
  askNewList(prompt: string): Promise<Task[]>;
  close(): Promise<void>;
}

// 1) API tabanlı (örnek şablon)
export class ApiCodexClient implements CodexClient {
  async submitTask(repoFullName: string, task: Task) {
    // TODO: Gerçek Codex API endpointini çağırın
    // fetch(`${CONFIG.codex.apiBase}/tasks`, { headers: { Authorization:
`Bearer ${CONFIG.codex.apiKey}` }, ... })
    console.log(`[CODEX/API] submit → ${repoFullName}: ${task.title}`);
    return { jobId: `api-${Date.now()}` };
  }
}

```

```

    }
    async waitForCompletion(jobId: string) {
        console.log(`[CODEX/API] wait → ${jobId}`);
        // TODO: job status polling
        await new Promise((r) => setTimeout(r, 3000));
        return 'success';
    }
    async createPR(opts: { title: string; body: string }) {
        console.log(`[CODEX/API] create pr → ${opts.title}`);
        // TODO: Codex üzerinden PR tetikleme destekliyse buraya ekleyin
    }
    async askNewList(prompt: string) {
        console.log(`[CODEX/API] ask → prompt(${prompt.slice(0, 40)}...)`);
        // TODO: model cevabını parse edip maddelere bölün
        return [
            { title: 'Örnek yeni görev 1' },
            { title: 'Örnek yeni görev 2' },
        ];
    }
    async close() {}
}

// 2) UI Otomasyonu (Playwright)
export class UiCodexClient implements CodexClient {
    private browser!: Browser;
    private page!: Page;

    async init() {
        this.browser = await chromium.launch({ headless: true });
        this.page = await this.browser.newPage();
        if (CONFIG.codex.cookiesJson) {
            try {
                const cookies = JSON.parse(await (await import('fs/promises')).readFile(CONFIG.codex.cookiesJson, 'utf8'));
                await this.page.context().addCookies(cookies);
            } catch {}
        }
        await this.page.goto(CONFIG.codex.uiUrl!);
        // TODO: Gerekirse login flow
        // await this.page.getByLabel('Email').fill(CONFIG.codex.uiEmail!);
        // await this.page.getByLabel('Password').fill(CONFIG.codex.uiPassword!);
        // await this.page.getByRole('button', { name: 'Sign in' }).click();
    }

    async submitTask(repoFullName: string, task: Task) {
        if (!this.page) await this.init();
        // TODO: Repo seçimi (örnek)
    }
}

```

```

    // await this.page.getByRole('combobox', { name: 'Repository' }).click();
    // await this.page.getByRole('option', { name: repoFullName }).click();

    // TODO: Görev/prompt alanına yazma ve gönderme
    // await this.page.getByPlaceholder('Describe the change').fill(task.title);
    // await this.page.getByRole('button', { name: 'Run' }).click();

    console.log(`[CODEX/UI] submit → ${repoFullName}: ${task.title}`);
    return { jobId: `ui-${Date.now()}` };
}

async waitForCompletion(jobId: string, opts?: { timeoutMs?: number }) {
    const timeout = opts?.timeoutMs ?? 10 * 60 * 1000; // 10 dakika
    const started = Date.now();
    console.log(`[CODEX/UI] wait → ${jobId}`);

    // TODO: UI'da başarı göstergesini bekleme (örnek selector)
    // await this.page.waitForSelector('text=Task completed successfully',
    { timeout });

    // Basit mock bekleme
    while (Date.now() - started < Math.min(3000, timeout)) {
        await new Promise((r) => setTimeout(r, 500));
    }
    return 'success';
}

async createPR(opts: { title: string; body: string }) {
    // TODO: Codex UI'da "create pr" butonu/komutu varsa onu tetikleyin
    // await this.page.getByRole('button', { name: 'Create PR' }).click();
    // await this.page.getByLabel('Title').fill(opts.title);
    // await this.page.getByLabel('Description').fill(opts.body);
    // await this.page.getByRole('button', { name: 'Submit' }).click();
    console.log(`[CODEX/UI] create pr → ${opts.title}`);
}

async askNewList(prompt: string) {
    // TODO: UI'da "ask" alanına prompt'u gönderip listeyi metinden parse edin
    console.log(`[CODEX/UI] ask → prompt(${prompt.slice(0, 40)}...)`);
    // Mock dönüş:
    return [
        { title: 'Yeni görev A' },
        { title: 'Yeni görev B' },
        { title: 'Yeni görev C' },
    ];
}

```



```

    async close() {
      await this.browser?.close();
    }
  }

export function createCodexClient(): CodexClient {
  if (CONFIG.codex.driver === 'api') return new ApiCodexClient();
  return new UiCodexClient();
}

```

## src/github.ts

```

import { Octokit } from '@octokit/rest';
import { CONFIG } from './config.js';

const octokit = new Octokit({ auth: process.env.GITHUB_TOKEN });

export async function createPrViaGithub(opts: { title: string; body: string;
branch: string }) {
  const { owner, repo, baseBranch } = CONFIG;

  // Branch zaten Codex tarafından oluşturulmuşsa bu adımı atlayabilirsiniz.
  // Burada örnek olarak boş bir branch oluşturuyoruz (commit yoksa PR açılmaz
  → gerçek akışta değişiklik içeren branch kullanın).

  // Base SHA al
  const baseRef = await octokit.repos.getBranch({ owner, repo, branch:
baseBranch });
  const baseSha = baseRef.data.commit.sha;

  // Yeni ref oluştur (refs/heads/branch)
  await octokit.git.createRef({ owner, repo, ref: `refs/heads/${opts.branch}`,
sha: baseSha });

  // PR aç
  const pr = await octokit.pulls.create({
    owner,
    repo,
    head: opts.branch,
    base: baseBranch,
    title: opts.title,
    body: opts.body,
  });
}

```

```
    return pr.data.html_url;
}
```

**Önemli:** Gerçek değişiklikleri Codex'in branch'e **push etmiş olması** idealdir. Eğer değişiklikler GitHub'a yansımıyorsa, PR anlamlı olmayacaktır. Bu durumda PR'ı Codex UI içinden tetiklemek (`PR_STRATEGY=codex`) daha doğru olur.

## src/index.ts

```
import { promises as fs } from 'fs';
import path from 'path';
import { marked } from 'marked';
import { CONFIG, assertConfig } from './config.js';
import { loadState, saveState } from './state.js';
import { createCodexClient, Task } from './codex.js';
import { createPrViaGithub } from './github.js';

async function readFileSafe(p: string) {
    return fs.readFile(p, 'utf8');
}

function parseTasks(md: string): { tasks: Task[]; checked: boolean[] } {
    const lines = md.split(/\r?\n/);
    const tasks: Task[] = [];
    const checked: boolean[] = [];
    for (const line of lines) {
        const m = line.match(/^s*- \[( |x|X)\] (.+)$/);
        if (m) {
            const done = m[1].toLowerCase() === 'x';
            tasks.push({ title: m[2].trim() });
            checked.push(done);
        }
    }
    return { tasks, checked };
}

async function markTaskDone(filePath: string, taskIndex: number) {
    const md = await readFileSafe(filePath);
    const lines = md.split(/\r?\n/);
    let idx = -1;
    for (let i = 0; i < lines.length; i++) {
        const m = lines[i].match(/^s*- \[( |x|X)\] (.+)$/);
        if (m) {
            idx++;
            if (idx === taskIndex) {

```

```

        lines[i] = lines[i].replace('- [ ]', '- [x]');
        break;
    }
}
}
await fs.writeFile(filePath, lines.join('\n'), 'utf8');
}

async function overwriteTasks(filePath: string, tasks: Task[]) {
    const body = tasks.map((t) => '- [ ] ${t.title}').join('\n');
    await fs.writeFile(filePath, body + '\n', 'utf8');
}

function batchBranchName(batchNumber: number) {
    const stamp = new Date().toISOString().replace(/[:.]/g, '-');
    return `codex/batch-${batchNumber}-${stamp}`;
}

async function main() {
    assertConfig();
    const state = await loadState();
    const codex = createCodexClient();

    const tasksFile = CONFIG.tasksFile;
    const tasksMd = await readFileSafe(tasksFile);
    const { tasks, checked } = parseTasks(tasksMd);

    // Sıradaki TODO bulun
    let i = state.nextTaskIndex;
    while (i < tasks.length && checked[i]) i++;

    if (i >= tasks.length) {
        // Liste bitti → ask ile yeni liste
        const prompt = await readFileSafe(CONFIG.newListPromptFile);
        const newList = await codex.askNewList(prompt);
        if (newList.length === 0) {
            console.log('Yeni liste boş döndü. Çıkılıyor.');
```

```

    }

    const task = tasks[i];
    console.log(`Görev gönderiliyor (${i + 1}): ${task.title}`);
    const job = await codex.submitTask(`${CONFIG.owner}/${CONFIG.repo}`, task);
    const res = await codex.waitForCompletion(job.jobId, { timeoutMs: 15 * 60 *
1000 });

    if (res === 'success') {
        await markTaskDone(tasksFile, i);
        state.totalCompleted += 1;
        state.successInBatch += 1;
        state.nextTaskIndex = i + 1; // bir sonrakine geç
        await saveState(state);
        console.log(`Görev başarıyla tamamlandı → [x] işaretlendi.`);
    } else {
        console.log(`Görev başarısız → tekrar deneme veya manuel müdahale
gerekli.`);
        await codex.close();
        return;
    }

    // Her 3 başarıda PR akışı
    if (state.successInBatch >= CONFIG.prEvery) {
        const title = `Batch #${state.batchNumber} - ${CONFIG.prEvery} görev
tamamlandı`;
        const body = `Bu PR, batch #${state.batchNumber} kapsamında tamamlanan $
${CONFIG.prEvery} görevi içerir.`;

        if (process.env.PR_STRATEGY === 'github') {
            const branch = batchBranchName(state.batchNumber);
            const url = await createPrViaGithub({ title, body, branch });
            console.log(`PR açıldı (GitHub API): ${url}`);
        } else {
            await codex.createPR({ title, body });
            console.log(`PR akışı Codex üzerinden tetiklendi.`);
        }

        state.successInBatch = 0;
        state.batchNumber += 1;
        await saveState(state);
    }

    await codex.close();
}

main().catch((e) => {

```

```
    console.error(e);
    process.exit(1);
  });
```

## GitHub Action (opsiyonel)

`.github/workflows/orchestrate.yml`

```
name: Codex Orchestrator

on:
  workflow_dispatch: {}
  schedule:
    - cron: '*/*/* * * * *' # her 30 dakikada bir (isteğe göre)

jobs:
  run:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 20
      - run: npm i -g pnpm
      - run: pnpm i
      - run: pnpm start
    env:
      GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
      GITHUB_OWNER: ${ vars.GITHUB_OWNER }
      GITHUB_REPO: ${ vars.GITHUB_REPO }
      BASE_BRANCH: ${ vars.BASE_BRANCH }
      PR_STRATEGY: ${ vars.PR_STRATEGY }
      CODEX_DRIVER: ${ vars.CODEX_DRIVER }
      CODEX_API_BASE: ${ secrets.CODEX_API_BASE }
      CODEX_API_KEY: ${ secrets.CODEX_API_KEY }
      CODEX_UI_URL: ${ vars.CODEX_UI_URL }
      CODEX_UI_EMAIL: ${ secrets.CODEX_UI_EMAIL }
      CODEX_UI_PASSWORD: ${ secrets.CODEX_UI_PASSWORD }
```

## Uygulama Notları

- **Gerçek Codex Entegrasyonu:** `src/codex.ts` içinde **API** veya **UI** sürücüsündeki `TODO` kısımlarına **gerçek endpointleri** (veya UI selector'larını) ekleyin. UI tarafında bir kez giriş yaptıktan sonra `cookies.json` kaydedip tekrar kullanabilirsiniz.
- **PR Stratejisi:** Değişiklikleri Codex zaten bir **branch**'e push ediyorsa, PR'ı Codex içinden tetiklemek daha sağlıklı. Aksi durumda GitHub API ile PR açarken **değişiklik içeren branch** kullanın.
- **Dayanıklılık:** Zaman aşımı, yeniden deneme ve hata yönetimini (ör. 3 denemeye kadar) ihtiyaca göre artırabilirsiniz.
- **Durum (state) dosyası** sürüm kontrolüne alınabilir; ancak CI'da artefact olarak tutmak daha temizdir.
- **Task parse** basit Markdown checkbox formatını kullanır. Başka kaynaklardan liste çekilecekse parser'ı genişletebilirsiniz.

## Sonuç

Bu iskelet, istediğiniz “**sıradaki görevi ver → başarılı olunca bir sonrakine geç → her üç başarıda PR → liste bitince ask ile yeni liste al ve döngüyü sürdür**” mantığını uçtan uca kurar. Sadece **Codex entegrasyonu** (API veya UI selector'ları) kısmını sisteme uygun şekilde doldurmanız yeterli.

## Mükemmel İnsan Dili Promptu (Codex Web + GitHub) — Final

**Giriş URL'si:** <https://chatgpt.com/codex>

**Repository'ler:** SOIPack, ch10gate, VideoKit-V10.10.2-copy-3

**Repo seçici (UI):** `.min-w-fit > div:nth-child(1) > div:nth-child(1) > div:nth-child(1) > div:nth-child(1) > div:nth-child(1) > button:nth-child(1)` → çekmece açılır, `90karatinsa/<repo>` seçilir.

**Code butonu (UI):** `.btn-primary` (doğrulanacak)

**Branch menüsü (UI):** `div.rounded-full:nth-child(2) > button:nth-child(1)`

### Kaynak Liste ve Format

- Görev kaynağı **tasks.md** (tek dosya). Başlıklar: `VideoKit`, `ch10gate`, `SOIPack`, her başlığın altında maddeler.
- Tamamlanma işareti: her maddenin solundaki parantez ( ) → boş; (+) → tamamlandı.
- Bir repo için tüm maddeler (+) olduğunda o repo **tamamlanmış** sayılır.
- Tamamlanan her repo için hemen altına yeni başlık açılır: `RepoName V2`, `RepoName V3`, ... ve yeni gelen maddeler bu başlığın altına ( ) ile eklenir.

### Döngü ve Sıra Kuralları

- “İşi biten repo”dan devam: sıra önemsiz; hangi repo bir işi bitirdiyse ona hemen yeni iş verilir.
- **Eşzamanlılık:** Aynı anda birden fazla repo iş alabilir; **aynı repo için eşzamanlı yalnızca 1 iş**.
- Her çalıştırmada **iki görev tek prompt içinde** verilir (format: `1. <görev> 2. <görev>`).

- Eğer hata tespit edilirse (aşağıya bak), yeniden denemede **tek görev** ( 1. <görev> ) verilir.

## Başarı / Hata Tespiti

- **Başarı kriteri:** sağ panelde  **diff göstergesi** görünür ve ilgili maddede ( ) → (+) dönüşümü yapılır.
- **Hata kriteri:** görev modu olmasına rağmen  **diff** hiç görünmüyorsa (ve *prompt üretme* modunda da değilse) → hata varsayılır.
- **Kontrol sıklığı:** her **60 saniyede** bir UI kontrol edilir (yüklenme/bekleme sonrası).

## Görev Verme Adımları (Normal Mod)

- 1) **Repo seç:** repo çekmecesini aç →  seç.
- 2) **Görev belirle:** ilgili reponun  içindeki **en güncel başlık** altında ( ) olan ilk iki maddeyi al.
- 3) **Prompt oluştur:** tek prompt içinde  formatıyla yaz.
- 4) **Çalıştır:** **Code** butonuna bas (  ).
- 5) **Takip:** her 60 sn'de bir  kontrol et. Başarıysa ilgili maddelerin parantezlerini (+) yap ve bir sonraki iki maddeye geç.
- 6) **Hata durumunda:**  çıkmadıysa veya "I can't do this" benzeri geri dönüş aldıysan → **tek madde** ile yeniden çalıştır (  ), ardından yine  kontrol et.

## "Prompt Üretme" Modu (Liste Bittiğinde)

- Bir reponun mevcut başlığı altındaki tüm maddeler (+) olduysa, o repo **liste bitti** sayılır.
- İlgili repo için **repo-özel** "prompt üretme promptu" dosyasını kullan: .
- Bu dosyada şu şablon bulunur (projeye özgü {PROJE\_TANIMI} yerleştirilmiş haldedir):

Rolün: Kıdemli görev kırıcı (task designer) + yazılım mimarı + test mühendisi.  
Erişimin: Kod reposunun tamamı (kaynak, test, örnek veri, CI/CD, dokümantasyon).  
Proje tanımı: "{PROJE\_TANIMI}". Görevin: Depoyu ve tırnak içindeki proje tanımını okuyup, endüstride kabul görmüş asgari (baseline) özellikleri içselleştir; depoyla kıyasla; eksikleri ve rakipleri aşmak için gereken farklılaştırıcıları belirle; ardından hepsini birbirinden bağımsız çalışabilecek yapay zekâ geliştiricilere verilecek şekilde "atomik görev/prompt"lara dönüştür.  
Çıktı kuralları: SOHBETE DÜZ METİN yaz; başlık/özet/yorum yok; sadece numaralı liste; 1 en kritik, numara büyüdükçe önem azalsın; toplam 8-15 madde üret. Her madde tek paragraf olsun, emir kipinde başla, mutlaka değişecek dosya ve/veya dizinleri belirt (mevcut değilse makul bir yol öner ve "varsayım" de), gerekli teknik adımları kısa ve net yaz, ilgili birim/integrasyon/performans/negatif testleri ve dokümantasyon güncellemelerini ekle (README/USER GUIDE/OpenAPI/CLI help/release notes), gerekiyorsa mig./paketleme adımlarını belirt; her maddeyi "accepted when ..." ile bitecek ölçülebilir kabul kriteriyle kapat.  
Önceliklendirme ilkeleri: kabul kapısını engelleyen doğruluk/güvenilirlik boşlukları, güvenlik/kripto ve veri bütünlüğü, performans/ölçek ve kaynak kullanımı, uyumluluk/raporlama artefaktları, paketleme/operasyon ve geliştirici

deneyimi. Soru sorma; makul varsayımlarını açıkça belirt. Yalnız listeyi yaz; başka hiçbir metin ekleme.

- Gelen yanıt **numaralı düz metin** olmalı. Bu maddeleri `tasks.md` dosyasına ilgili reponun bir altına **yeni başlık** olarak ekle (`RepoName V<n>`), her maddenin soluna ( ) koy. Sonra **Normal Mod** adımlarına geri dön.

### PR / Branch Döngüsü (Her 3 Başarılı İş Sonrası)

1) Aynı repo içinde **ardışık 3 başarılı görev** tamamlandığında **Create PR** akışını başlat.

2) GitHub'da **confirm merge** yap.

3) Codex UI'a dön → **branch menüsü** `div.rounded-full:nth-child(2) > button:nth-child(1)` butonuna bas → açılan aramada **son oluşturulan branch** adını gir.

- Branch adı kaynağı: GitHub'daki **PR başlığı** (veya Codex'te görünen son görev başlığı).

4) Son branch'i seç ve **kaldığın yerden bir sonraki görevi** ver. `Code` butonuna bas.

**Notlar:** - Branch oluşturma/push işlemleri büyük olasılıkla Codex tarafından yapılır (**varsayım**). PR akışında branch adını GitHub/PR görünümünden al.

- UI tarafında `+/-` diff **sadece normal görevler** için beklenir; *prompt üretme* adımlarında diff beklenmez.

### Zamanlama ve Dayanıklılık

- Her görev çalıştırmadan sonra **60 sn** aralıklarla UI kontrolü yapılır.
- Görevler **repo başına tek akış**, repo'lar arası **paralel** ilerleyebilir.
- Oturumun düşmeyeceği varsayılır; yetkilendirme yeniden denemesi yapılmaz.

### Kısa Özet

- Repolar: **SOIPack, ch10gate, VideoKit-V10.10.2-copy-3**
- **İki görev** → **Code** → `+/-` kontrol → başarılıysa (+) işaretle ve ilerle; hata varsa **tek göreve** düşür.
- Liste bitince **repo-özel prompt** ile **yeni liste** üret, `RepoName V<n>` ile ekle.
- Her **3 başarılı**da **PR aç + merge**, sonra **branch seç** ve devam et.