RepoArchitectAgent

Product Requirements Document (PRD)

Project name

RepoArchitectAgent — Instant repo architecture diagram, stack explanation, and CI/CD generator.

One-line pitch

Paste any public GitHub repo URL and get an interactive architecture diagram, prioritized onboarding + risk report (LLM-generated), and an auto-opened PR with a generated CI pipeline — all in two clicks.

Target users

- New contributors to open-source projects
- Maintainers who want quick audits / onboarding
- DevOps/engineering leads assessing repo health
- Hackathon judges (sponsorship alignment)

Problem statement

Onboarding and initial assessment of unfamiliar repos is slow and error prone. Developers waste time understanding structure, discovering hotspots (missing tests, huge files, insecure deps), and setting up CI/CD. A tool that generates visual architecture, explains the stack, and produces a ready CI pipeline saves hours and improves code quality.

Objectives (success metrics)

- **Demonstrable MVP**: Visual diagram + textual explanation + opened PR with CI for two sample repos within 48 hours.

- **Sponsor integration**: Show usage artifacts for Cline, Kestra, Oumi, CodeRabbit, and Vercel.
- **Usability**: A judge can run the demo in under 2 minutes using the Vercel UI and see the PR in the sample repo.
- **Quality**: PR must include clear explanation and be non-destructive (pipeline only).

Core MVP features (must-have)

1. Web UI (deployed on Vercel) with a single input for GitHub repo URL and a start button.
2. Backend analysis pipeline: clone repo (shallow), extract repo shape (language, files, dependencies, endpoints).
3. Mermaid (or SVG) architecture diagram generated from extraction.
4. LLM-generated textual outputs: high-level summary, top 5 hotspots, 3-step onboarding checklist.
5. CI generator that writes a minimal GitHub Actions YAML, commits to a branch, and opens a PR.
6. Evidence of sponsor tech usage:
    - Cline commands or logs (or documented substitution) used to perform cloning/PR actions.
    - Kestra blueprint that orchestrates steps (clone → analyze → diagram → generate CI → open PR).
    - Oumi prompt/response artifacts used to generate text (or a documented LLM fallback).
    - CodeRabbit config in repo showing how review would run on PR.
    - Vercel deployed frontend and preview URL.

Stretch features (if time permits)

- PR contains additional artifacts: README improvements, `.env.example`, basic test skeleton.

- Risk heatmap overlay on diagram (color nodes).
- Automatic validation run via Kestra after PR, showing lint success/fail.
- Browser extension / GitHub UI button (V1).
- Small unit testing & CI caching.

Non-functional requirements

- Response time for analysis on demo repos: < 90 seconds (for demo).
- All generated PRs non-destructive and clearly labelled.
- Minimal secrets usage: GitHub token stored in environment for CI actions, no user tokens saved.
- Easy reproduce: single README command to run end-to-end locally.

Constraints & assumptions

- Two-day timeline: no heavy model fine-tuning.
- Use shallow git clone and heuristics for analysis to be fast.
- If Oumi access is limited, use a high-quality hosted LLM as fallback and document swap plan.

Risks & mitigation

- **Risk:** PR blocking (bad pipeline). → **Mitigate:** Make pipeline tolerant: run commands with fallbacks (`|| echo "no-tests"`).
- **Risk:** Sponsor tool integration not fully available. → **Mitigate:** Provide documented fallback and captured logs that show the intended integration.
- **Risk:** Cloning large repos slow. → **Mitigate:** `-depth 1` and use sample small repos for demo.

Acceptance criteria

- Paste a GitHub URL → diagram renders + text explanation appears + button to generate CI opens a PR that appears in the target repo within the demo.

- Kestra blueprint and Cline command artifacts in repo.
- Vercel www preview accessible and included in submission.
- README and demo video (≤ 2 minutes) available.

Architecture and Data Flow

1. **Frontend (Vercel / Next.js)**
    - UI page with repo input, progress UI, Mermaid diagram canvas, textual panel, PR link.
2. **API layer (Next.js serverless / Node)**
    - `/api/analyze` accepts `repoUrl`, triggers Kestra orchestration (or local worker) and returns status + outputs.
3. **Analysis worker(s)**
    - `api/analyze_repo.py` (Python): clones repo (depth 1), extracts languages, files, dependencies, endpoints, build config.
    - `api/generate_mermaid.js`: converts JSON shape → Mermaid markup.
    - `api/generate_ci.js`: returns CI YAML content.
    - `api/open_pr.js`: uses GitHub REST API / `gh` CLI or Cline to commit branch + open PR.
4. **Orchestration (Kestra)**
    - Blueprint controls steps: clone → analyze → diagram → generate CI → open PR → notify.
5. **LLM (Oumi or fallback)**
    - Prompts run on extracted repo shape to produce summary, hotspots, onboarding checklist. Prompts & responses saved.
6. **Cline**

- Use `cline` for at least one scripted operation (e.g., `cline run open-pr`) or include its CLI logs as evidence.

7. **CodeRabbit**
   - Add `.coderabbit.yml` to show auto-review configuration for opened PRs.
8. **Persisted artifacts**
   - Store run logs & outputs in `runs/` or a lightweight DB (SQLite) for demo reproducibility.

Folder structure

```
repo-architect-agent/
├─ .github/
│  └─ workflows/
│     └─ deploy.yml              # CI to deploy demo to Vercel (optional)
├─ kestra/
│  └─ blueprint_repo_analysis.yml   # Kestra blueprint
├─ api/
│  ├─ analyze_repo.py            # Repo analysis (Python)
│  ├─ generate_mermaid.js        # Mermaid generator (Node)
│  ├─ generate_ci.js             # CI YAML generator (Node)
│  ├─ open_pr.js                 # Open PR via GH REST / gh / Cline
│  └─ helpers/
│     └─ repo_parsers.py         # helper parsers (Python)
├─ web/
│  ├─ package.json
│  ├─ next.config.js
│  ├─ pages/
│  │  ├─ index.js                # main UI
│  │  └─ api/
│  │     └─ analyze.js           # /api/analyze route
│  ├─ components/
```

```
│   │   └── MermaidViewer.jsx
│   └── public/
├── docs/
│   ├── PRD.md
│   ├── DEMO.md
│   └── OUMI_PROMPTS.md            # saved LLM prompts & responses
├── runs/                          # run artifacts produced during demo
│   └── <timestamp>/
│       ├── repo_shape.json
│       ├── diagram.mmd
│       └── generated-ci.yml
├── .coderabbit.yml                # CodeRabbit config sample
├── README.md
└── LICENSE
```

Tech Stack

- Frontend / UI: **Next.js** (React) — deployed on **Vercel**
- Diagram: **Mermaid** (client side)
- Backend: **Node.js** serverless (Next API) + small **Python** analysis api
- Orchestration: **Kestra** (blueprints)
- CLI automation / dev ops: **Cline** (CLI commands), **gh** CLI (fallback)
- Code review automation: **CodeRabbit** (config)
- LLM / reasoning: **Oumi** (preferred) — or fallback: OpenAI or other hosted LLM; save all prompts/responses
- GitHub interactions: GitHub REST API or `gh` CLI
- CI for generated pipelines: **GitHub Actions**
- Data storage for runs: lightweight **SQLite** or filesystem `runs/`

- Container (optional dev): **Docker**
- Logging/observability: console logs saved to runs and display screenshots in demo

---

README

README.md

````
# RepoArchitectAgent

**Instant repo architecture diagram, stack intelligence, and CI generator.**

## What it does
Paste a public GitHub repo URL and RepoArchitectAgent will:
1. Clone the repo (shallow) and analyze its structure.
2. Generate an interactive architecture diagram (Mermaid).
3. Produce an LLM-generated summary: high-level architecture, top hotspots, and a 3-step onboarding checklist.
4. Generate a minimal GitHub Actions CI pipeline, open an automated PR with the pipeline, and provide a PR link.
5. Store orchestration blueprint (Kestra) and show evidence of sponsor tool us age (Cline, Oumi, CodeRabbit).

## Demo
- Live demo (Vercel): `https://<your-vercel-deployment>.vercel.app`
- Demo video (2 minutes): `LINK_TO_VIDEO`

## Quick start (local)
1. Clone:
  ```bash
````
````

```
git clone https://github.com/<your-org>/repo-architect-agent.git
cd repo-architect-agent/web
```

2. Install

```
cd web
npm install
cd ..
pip3 install -r requirements.txt   # for api/
```

3. Setup Environment (`.env.local` in `web/`)

```
GITHUB_TOKEN=ghp_xxx        # token with repo permissions (for demo PRs)
OUMI_API_KEY=xxx            # or use OPENAI_KEY as fallback
```

4. Run development server

```
cd web
npm run dev
```

5. In the UI, paste a public GitHub repo URL and press Analyze.

Files and Key Locations

- `kestra/blueprint_repo_analysis.yml` — orchestration blueprint
- `api/analyze_repo.py` — repo analyzer
- `api/generate_ci.js` — CI generator
- `api/open_pr.js` — PR opener
- `.coderabbit.yml` — CodeRabbit config sample

- `docs/OUMI_PROMPTS.md` — stored prompts & responses for reproducibility

How it uses sponsor tools

- **Cline** — referenced in README and used to run scripted operations (`cline run demo:open-pr`). See `docs/ClineUsage.md`.
- **Kestra** — blueprint in `kestra/` demonstrates orchestration (clone → analyze → generate → open-pr).
- **Oumi** — prompts saved in `docs/OUMI_PROMPTS.md` used to produce summaries and hotspot rankings.
- **CodeRabbit** — `.coderabbit.yml` included to show auto-review rules for generated PRs.
- **Vercel** — app is designed for Vercel deployment; `vercel.json` included.

Notes and Safety

- Generated PRs are non-destructive and labeled clearly. Review before merging.
- For large repos, the shallow clone keeps demo time low.
- If Oumi keys are not available, the project falls back to an alternate LLM (documented in `docs/`).

Full Length Prompt

I need you to scaffold a production-ready hackathon project called "RepoArchitectAgent". Create a Next.js (React) frontend deployed on Vercel and serverless API endpoints that implement an analysis pipeline for GitHub repos. The project must include:
- A web UI (`web/`) with a simple form to input a public GitHub repo URL, progress steps, and a Mermaid-based diagram viewer for architecture diagrams.
- An `/api/analyze` endpoint that kicks off a background orchestration. Provide code to call a local Kestra blueprint (YAML) or call orchestration functions that run the following steps: shallow clone `git clone --depth 1`, extract repo shape (language detection, top-level dirs, package.json or requirements.txt dependencies, presence of Dockerfile, list API endpoints like Next.js pages/api, Express routes, FastAPI), save `repo_shape.json`.
- A Python script `api/analyze_repo.py` that performs repo cloning and produces `repo_shape.json`.
- A Node script `api/generate_mermaid.js` that converts `repo_shape.json` into a Mermaid `graph` and outputs `diagram.mmd`.
- A Node script `api/generate_ci.js` that generates a minimal GitHub Actions YAML based on repo type: Node (npm ci, lint, test, build), Python (pip install -r requirements.txt, pytest), Next.js (npm build, vercel preview). Save YAML to `.github/workflows/ci-generated.yml`.
- A script `api/open_pr.js` that creates a branch, commits the generated CI YAML, pushes, and opens a PR using the GitHub REST API or `gh` CLI. Provide fallback using documented Cline CLI command `cline run` (create a wrapper so it can be swapped).
- A Kestra blueprint `kestra/blueprint_repo_analysis.yml` that orchestrates clone → analyze → diagram → generate_ci → open_pr steps (shell tasks referencing the api).
- Add a `.coderabbit.yml` sample for CodeRabbit that demonstrates lint/test enforcement on PRs.
- Integrate an LLM step: create `docs/OUMI_PROMPTS.md` and code that calls Oumi (if Oumi API unavailable, default to OpenAI) to produce a 3-sentence high-level summary, top-5 hotspots, and a 3-step onboarding checklist from `repo_shape.json`. Save prompts and responses to `docs/`.
- Add `runs/` folder to save run artifacts (`repo_shape.json`, `diagram.mmd`, `generated-ci.yml`, LLM responses, PR link).

- Provide a simple local dev flow and a Vercel deployment configuration (`vercel.json` if needed).
- Provide a sample GitHub Actions YAML for the generated pipelines and a separate `deploy.yml` to deploy the frontend to Vercel on push to main.
- Create a professional `README.md`, `PRD.md`, and `DEMO.md` that explains the architecture, how to reproduce the demo in 3 steps (run local server, paste repo URL, generate CI and click PR link), and where to find sponsor tool usage evidence.
- Implement a minimal, working UI that demonstrates the flow end-to-end on two small sample repos (one Node and one Python). Use shallow clones to keep run times short.
- For sponsor tools, produce artifacts demonstrating usage:
    - **Cline**: include example commands in `docs/ClineUsage.md` and a wrapper script `api/run_cline.sh` that either runs real `cline` or simulates behavior and logs output.
    - **Kestra**: commit the blueprint and include a note how to run it locally (kestra CLI details).
    - **Oumi**: store prompts/responses and mark where Oumi is used; include a fallback to OpenAI if needed.
    - **CodeRabbit**: include `.coderabbit.yml` and instructions how PR comments will be generated.
    - **Vercel**: include deployment notes and env var names used (GITHUB_TOKEN, OUMI_API_KEY).
- Keep the generated CI tolerant so demo PRs do not block (use `|| echo "no-tests"` fallbacks).
- Create `api/demo.sh` that runs a full demo using a sample repo, saves artifacts to `runs/` and prints the PR URL.
- Provide initial unit test placeholders and a `DEMO.md` with a 2-minute demo script (what to say and click).
- Ensure all prompts, API calls, and integration points are saved as files in `docs/` for judge verification.

Output: generate complete file scaffolding and fill out the files with production-ready starter code, including `web/pages/index.js`, `web/pages/api/analyze.js`, `api/analyze_repo.py`, `api/generate_mermaid.js`, `api/generate_ci.js`, `api/open_pr.js`, `kestra/blueprint_repo_analysis.yml`, `.coderabbit.yml`, `README.

md`, `PRD.md`, `DEMO.md`, and `docs/OUMI_PROMPTS.md`. Make the code r
unnable locally with minimal configuration and document all environment varia
bles and steps clearly in README.

> **Use this exact prompt with GitHub Copilot / Copilot Chat / an AI assistant*
*. It instructs the assistant to scaffold the full project, create the files above, an
d wire sample integrations. Copy-paste the block below into Copilot (or reuse
sentences as necessary).
>
> ```
> I need you to scaffold a production-ready hackathon project called "RepoAr
chitectAgent". Create a Next.js (React) frontend deployed on Vercel and serve
rless API endpoints that implement an analysis pipeline for GitHub repos. The
project must include:
> - A web UI (`web/`) with a simple form to input a public GitHub repo URL, pr
ogress steps, and a Mermaid-based diagram viewer for architecture diagram
s.
> - An `/api/analyze` endpoint that kicks off a background orchestration. Provi
de code to call a local Kestra blueprint (YAML) or call orchestration functions t
hat run the following steps: shallow clone `git clone --depth 1`, extract repo sh
ape (language detection, top-level dirs, package.json or requirements.txt dep
endencies, presence of Dockerfile, list API endpoints like Next.js pages/api, E
xpress routes, FastAPI), save `repo_shape.json`.
> - A Python script `api/analyze_repo.py` that performs repo cloning and prod
uces `repo_shape.json`.
> - A Node script `api/generate_mermaid.js` that converts `repo_shape.json` i
nto a Mermaid `graph` and outputs `diagram.mmd`.
> - A Node script `api/generate_ci.js` that generates a minimal GitHub Actions
YAML based on repo type: Node (npm ci, lint, test, build), Python (pip install -r
requirements.txt, pytest), Next.js (npm build, vercel preview). Save YAML to `.
github/workflows/ci-generated.yml`.
> - A script `api/open_pr.js` that creates a branch, commits the generated CI Y
AML, pushes, and opens a PR using the GitHub REST API or `gh` CLI. Provide f
allback using documented Cline CLI command `cline run` (create a wrapper so

it can be swapped).
> - A Kestra blueprint `kestra/blueprint_repo_analysis.yml` that orchestrates cl one → analyze → diagram → generate_ci → open_pr steps (shell tasks referen cing the api).
> - Add a `.coderabbit.yml` sample for CodeRabbit that demonstrates lint/test enforcement on PRs.
> - Integrate an LLM step: create `docs/OUMI_PROMPTS.md` and code that c alls Oumi (if Oumi API unavailable, default to OpenAI) to produce a 3-sentence high-level summary, top-5 hotspots, and a 3-step onboarding checklist from ` repo_shape.json`. Save prompts and responses to `docs/`.
> - Add `runs/` folder to save run artifacts (`repo_shape.json`, `diagram.mmd`, `generated-ci.yml`, LLM responses, PR link).
> - Provide a simple local dev flow and a Vercel deployment configuration (`ve rcel.json` if needed).
> - Provide a sample GitHub Actions YAML for the generated pipelines and a s eparate `deploy.yml` to deploy the frontend to Vercel on push to main.
> - Create a professional `README.md`, `PRD.md`, and `DEMO.md` that explai ns the architecture, how to reproduce the demo in 3 steps (run local server, p aste repo URL, generate CI and click PR link), and where to find sponsor tool u sage evidence.
> - Implement a minimal, working UI that demonstrates the flow end-to-end o n two small sample repos (one Node and one Python). Use shallow clones to k eep run times short.
> - For sponsor tools, produce artifacts demonstrating usage:
> - **Cline**: include example commands in `docs/ClineUsage.md` and a w rapper script `api/run_cline.sh` that either runs real `cline` or simulates behavi or and logs output.
> - **Kestra**: commit the blueprint and include a note how to run it locally (kestra CLI details).
> - **Oumi**: store prompts/responses and mark where Oumi is used; inclu de a fallback to OpenAI if needed.
> - **CodeRabbit**: include `.coderabbit.yml` and instructions how PR com ments will be generated.
> - **Vercel**: include deployment notes and env var names used (GITHUB _TOKEN, OUMI_API_KEY).
> - Keep the generated CI tolerant so demo PRs do not block (use `|| echo "no

-tests"` fallbacks).
> - Create `api/demo.sh` that runs a full demo using a sample repo, saves artifacts to `runs/` and prints the PR URL.
> - Provide initial unit test placeholders and a `DEMO.md` with a 2-minute demo script (what to say and click).
> - Ensure all prompts, API calls, and integration points are saved as files in `docs/` for judge verification.
>
> Output: generate complete file scaffolding and fill out the files with production-ready starter code, including `web/pages/index.js`, `web/pages/api/analyze.js`, `api/analyze_repo.py`, `api/generate_mermaid.js`, `api/generate_ci.js`, `api/open_pr.js`, `kestra/blueprint_repo_analysis.yml`, `.coderabbit.yml`, `README.md`, `PRD.md`, `DEMO.md`, and `docs/OUMI_PROMPTS.md`. Make the code runnable locally with minimal configuration and document all environment variables and steps clearly in README.
> ```

---

7) Step-wise, day-wise **2-day concise plan** (exact order, prioritized — copy/paste to run)

> **General guidance:** split team tasks so at least two people work in parallel (frontend + api). Keep scope small: diagram + summary + PR for 2 sample repos.

Day 0 (prep, optional, before 48-hour window)
- Reserve two demo repos (small public repos) and add their URLs to `docs/SAMPLE_REPOS.md`.
- Create GitHub token with `repo` permissions.
- Create Vercel account & project.

Day 1 — Core pipeline (hours 0–12)
1. Scaffold repo & Next.js app (`web/`).
2. Implement UI: input, progress, MermaidViewer, placeholder outputs.
3. Implement `api/analyze_repo.py`:

```
   - `git clone --depth 1 <repo> repo/`
   - detect language & framework, list files, parse package.json/requirements.
   - write `repo_shape.json`.
4. Implement `api/generate_mermaid.js` to convert `repo_shape.json` → `diagr
am.mmd`.
5. Hook `/api/analyze` to call `analyze_repo.py` and return `repo_shape.json` a
nd `diagram.mmd`.
6. Render Mermaid in UI.

**Checkpoint:** Paste sample repo → see diagram & repo_shape in UI.

## Day 1 — Afternoon/evening (hours 12–24)
7. Implement `api/generate_ci.js`:
   - Node / Python / Next.js templates minimal.
   - Save to `.github/workflows/ci-generated.yml`.
8. Implement `api/open_pr.js`:
   - Use `gh` CLI or GitHub REST to create branch, commit, push, open PR.
   - PR body template should include explanation and link back to demo.
9. Wire a "Generate CI & Open PR" button in UI that triggers CI generation and
runs `open_pr.js`.
10. Add `.coderabbit.yml` sample to repo and document CodeRabbit usage.

**Checkpoint:** Press "Generate CI" → PR opened in target repo.

## Day 2 — Orchestration, LLM, Kestra, polish (hours 24–36)
11. Create Kestra blueprint `kestra/blueprint_repo_analysis.yml` orchestrating s
hell tasks calling your api.
12. Add Oumi/OpenAI integration to generate the textual summary:
    - Implement `api/generate_summary.js` (call Oumi/OpenAI with `repo_shap
e.json` prompt).
    - Save prompt + response to `docs/OUMI_PROMPTS.md`.
13. Connect the summary output to the UI.

**Checkpoint:** UI shows diagram + LLM summary + PR link.

## Day 2 — Finishing touches & demo (hours 36–48)
```

14. Add `docs/`, `README.md`, `PRD.md`, `DEMO.md` — ensure clear reproduction steps.
15. Deploy UI to Vercel. Add vercel preview link to README.
16. Run demo twice on both sample repos — capture logs & save runs to `runs/`.
17. Create 2-minute demo video showing: paste repo, view diagram, click generate CI, show PR, mention Kestra / Cline / CodeRabbit artifacts in repo.
18. Final polish: screenshot of Kestra run (if run), Cline command logs, Oumi prompt file.

Deliverables by end of Day 2:
- Repo with code & blueprint, deployed Vercel URL, 2 PRs in sample repos, demo video, README/PRD/DEMO docs.

8) Minimal example files & snippets (copyable)

Example `api/analyze_repo.py` (very short stub)
```python
#!/usr/bin/env python3
# api/analyze_repo.py
import sys, json, subprocess, os
from pathlib import Path

def shallow_clone(repo_url, out='repo'):
    if os.path.exists(out):
        subprocess.run(['rm','-rf',out])
    subprocess.run(['git','clone','--depth','1',repo_url,out], check=True)

def detect_shape(path='repo'):
    shape = {'path': path, 'languages': [], 'files': [], 'frameworks': [], 'dependencies': []}
    for root, dirs, files in os.walk(path):
        for f in files:
            if f.endswith('.py'):
                shape['languages'].append('python')
```

```
            if f == 'package.json':
                shape['languages'].append('node')
                try:
                    import json as j
                    pj = j.load(open(os.path.join(root,f)))
                    shape['dependencies'].extend(list(pj.get('dependencies', {}).keys
()))
                except Exception:
                    pass
            shape['files'].append(os.path.relpath(os.path.join(root,f), path))
    shape['languages'] = list(set(shape['languages']))
    return shape

if __name__ == '__main__':
    repo = sys.argv[1]
    outdir = sys.argv[2] if len(sys.argv)>2 else 'runs/latest'
    os.makedirs(outdir, exist_ok=True)
    shallow_clone(repo, 'temp_repo')
    shape = detect_shape('temp_repo')
    with open(os.path.join(outdir,'repo_shape.json'),'w') as fh:
        json.dump(shape, fh, indent=2)
    print('Wrote', os.path.join(outdir,'repo_shape.json'))
```

Example

api/generate_mermaid.js

```
// node api/generate_mermaid.js repo_shape.json > diagram.mmd
const fs = require('fs');
const shape = JSON.parse(fs.readFileSync(process.argv[2] || 'runs/latest/rep
o_shape.json'));
let mermaid = 'graph TD\n';
if (shape.languages.includes('node')) mermaid += 'subgraph NodeJS\\nend
```

```
\n';
mermaid += `A[Repository] --> B[Languages: ${shape.languages.join(', ')}]\n`;
mermaid += `A --> C[Files: ${shape.files.slice(0,10).join(', ')}]\n`;
fs.writeFileSync('runs/latest/diagram.mmd', mermaid);
console.log('diagram generated at runs/latest/diagram.mmd');
```

api/generate_ci.js **(Node minimal)**

```
const fs = require('fs');
const shape = JSON.parse(fs.readFileSync(process.argv[2] || 'runs/latest/rep
o_shape.json'));
let ci = `
name: CI - Generated
on: [push, pull_request]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
`;
if (shape.languages.includes('node')) {
  ci += `
      - name: Setup Node
        uses: actions/setup-node@v4
        with: { node-version: '18' }
      - name: Install
        run: npm ci || echo "no-install"
      - name: Lint
        run: npm run lint || echo "no-lint"
      - name: Test
        run: npm test || echo "no-tests"
`;
} else {
  ci += `
```

```
    - name: Fallback step
      run: echo "No Node detected; please customize CI"
`;
}
fs.writeFileSync('.github/workflows/ci-generated.yml', ci);
console.log('CI written to .github/workflows/ci-generated.yml');
```

`api/open_pr.js` (Node, using gh)

```
// Requires GH CLI and GITHUB_TOKEN
const { execSync } = require('child_process');
const branch = 'agent/ci-generated';
try {
  execSync(`git checkout -b ${branch}`, { stdio: 'inherit' });
  execSync('git add .github/workflows/ci-generated.yml', { stdio: 'inherit' });
  execSync(`git commit -m "chore: add generated CI pipeline (agent)"`, { stdio:
'inherit' });
  execSync(`git push origin ${branch}`, { stdio: 'inherit' });
  execSync(`gh pr create --title "Add generated CI pipeline (agent)" --body "A
utomated CI generated by RepoArchitectAgent." --base main`, { stdio: 'inherit'
});
  console.log('PR created');
} catch (e) {
  console.error('PR creation failed', e.message);
  process.exit(1);
}
```

Note: For production, `open_pr.js` should interact with the target repo (clone target, create branch, push to that target remote). The above sample assumes you are running inside a fork/clone you can push to.

Final checklist before you submit

- Two sample public repos analyzed, PRs opened (public).
- Vercel URL included in README.
- Kestra blueprint committed; at least one run log saved.
- Cline usage documented and at least one command shown in logs.
- Oumi prompts & responses saved (or fallback documented).
- CodeRabbit config present and at least one sample review comment saved.
- Demo video ≤ 2 minutes uploaded and linked in README.
- `runs/` contains artifacts for judge verification.