**Goal**

Build an **agentic system** that, given requirements/user stories and a few selections (tools, languages, browsers, platforms, environments, test types, CI/CD, and report format), **plans, scaffolds, and runs** a complete test‑automation solution for Web UI, API, and backend services.

**What this agent can do**

* **Ingest**: Jira/ADO stories, Gherkin, OpenAPI/Swagger, Postman collections, DB schemas.
* **Plan**: Choose stacks/frameworks (POM, data‑driven, BDD), language (TypeScript/Java/Python), runners, browsers/platforms.
* **Scaffold**: Generate a repo (mono or multi), with ready‑to‑run UI/API/Backend projects.
* **Wire CI/CD**: Emit Jenkinsfile, Azure Pipelines YAML, AWS CodeBuild/CodePipeline, Google Cloud Build.
* **Generate tests**: From user stories (functional/smoke/regression) and from OpenAPI specs for APIs.
* **Manage configs**: Environments (dev/qa/stage/prod), secrets, test data.
* **Execute**: Local/Docker/Cloud grid, parallelization, retries, flake control.
* **Report**: Allure/Playwright HTML/Extent/ReportPortal; publish artifacts to CI/CD.

**High‑level architecture**

+-------------------+ +---------------------+ +---------------------+

| Requirements | | Tech‑Stack Planner | | Scaffolder |

| Ingestion Agent | ---> | (Rule Engine + | --> | (Jinja templates + |

| (LLM + parsers) | | LLM rationale) | | project writers) |

+-------------------+ +---------------------+ +---------------------+

| | |

v v v

+-------------------+ +---------------------+ +---------------------+

| Test Generator | | CI/CD Writer | | Runner & Orchestr. |

| (Gherkin/API/UI) | | (Jenkins/Azure/... )| | (CLI + schedulers) |

+-------------------+ +---------------------+ +---------------------+

| | |

v v v

+-------------------+ +---------------------+ +---------------------+

| Data & Secrets | | Env Config Manager | | Reporting Adapter |

| Manager (Vault) | | (.env/yaml/params) | | (Allure/Extent/...) |

+-------------------+ +---------------------+ +---------------------+

**Implementation baseline**

* Orchestrator in **Python** (fast templating + cross‑platform): Typer/Click, Pydantic, Jinja2, ruamel.yaml, cookiecutter‑style writing.
* LLM bridge (optional but powerful) for converting stories → tests and picking stacks.
* UI: **Playwright (TypeScript)** with POM (default) or Screenplay; Selenium (Java) optional adapter.
* API: **RestAssured (Java)** or **Playwright API (TS)** or **pytest + requests (Python)**.
* Backend/Integration: **pytest** (Python) with DB/queue/service hooks.
* Reporting: **Allure** as a common denominator (+ Playwright HTML, Extent, Surefire, ReportPortal optional).
* Packaging: **Monorepo** (pnpm + Maven + pip) or split repos.

**Agent contract (solution spec)**

**solution.yaml** – single source of truth. The agent reads this and does everything.

solution:

name: acme‑banking‑qa

description: Web + API + backend automation for Acme Banking

repo\_layout: monorepo # monorepo|polyrepo

languages:

ui: typescript

api: java

backend: python

ui:

framework: playwright

pattern: POM # POM|Screenplay

browsers: [chromium, firefox, webkit]

platforms: [windows, linux]

headless: true

api:

framework: restassured

inputs:

openapi: specs/banking.yaml

postman: specs/postman\_collection.json

backend:

framework: pytest

services: [db, queue]

test\_types: [functional, smoke, regression]

data\_strategy:

type: datadriven # datadriven|factory|hybrid

sources: [csv, json]

environments: [dev, qa, stage]

secrets:

provider: env # env|vault|aws‑secrets|azure‑kv|gcp‑sm

cicd:

providers: [jenkins, azure, aws, gcp]

default: jenkins

reporting:

providers: [allure, playwright\_html]

quality:

flaky\_retries: 2

parallelism: auto

Tip: Allow **overrides** via CLI (e.g., --ui.framework=selenium) and per‑env files config.qa.yaml.

**Decision matrix (simplified)**

|  |  |  |
| --- | --- | --- |
| **Need** | **Default Pick** | **Alternatives** |
| Cross‑browser UI on Windows/Linux/Mac | Playwright (TS) | Selenium (Java) when legacy grid/IE/Safari tech needed |
| API from OpenAPI | RestAssured (Java) codegen | Playwright API (TS), pytest+requests |
| Backend svc checks (DB, Kafka, cron) | pytest (Python) | Testcontainers Java/Python for envs |
| One report everywhere | Allure | Extent (Java), ReportPortal (server) |
| Cloud CI portable | Jenkinsfile + Azure YAML | GitHub Actions, CodeBuild/CloudBuild |

Rule engine can be a small YAML (editable) the Planner interprets.

**Repository layout (monorepo)**

acme‑banking‑qa/

solution.yaml

tools/agent/ # Python orchestrator

ui/playwright/ # TypeScript UI tests

api/restassured/ # Java API tests

backend/pytest/ # Python backend/integration

ci/

Jenkinsfile

azure‑pipelines.yaml

aws/buildspec.yml

gcp/cloudbuild.yaml

env/

config.dev.yaml

config.qa.yaml

config.stage.yaml

data/

users.csv

reports/.gitkeep

docs/README.md

**Orchestrator (Python) – skeleton**

# tools/agent/main.py

import json, shutil, subprocess, sys

from pathlib import Path

import typer

from pydantic import BaseModel

from ruamel.yaml import YAML

from jinja2 import Environment, FileSystemLoader

app = typer.Typer()

yaml = YAML()

tpl = Environment(loader=FileSystemLoader(str(Path(\_\_file\_\_).parent / "templates")), trim\_blocks=True, lstrip\_blocks=True)

class Solution(BaseModel):

solution: dict

ROOT = Path(\_\_file\_\_).resolve().parents[2]

def write(path: Path, content: str):

path.parent.mkdir(parents=True, exist\_ok=True)

path.write\_text(content, encoding="utf-8")

@app.command()

def plan(spec: str = "solution.yaml"):

data = yaml.load(Path(ROOT / spec).read\_text())

s = Solution(\*\*data)

typer.echo(json.dumps({"plan": s.solution}, indent=2))

@app.command()

def scaffold(spec: str = "solution.yaml"):

data = yaml.load(Path(ROOT / spec).read\_text())

s = Solution(\*\*data)

# UI

if s.solution['ui']['framework'] == 'playwright':

for f in ["package.json", "playwright.config.ts", "src/pages/LoginPage.ts", "tests/login.spec.ts"]:

template = tpl.get\_template(f+".j2")

write(ROOT/"ui"/"playwright"/f, template.render(sol=s.solution))

# API

if s.solution['api']['framework'] == 'restassured':

for f in ["pom.xml", "src/test/java/base/ApiTest.java", "src/test/java/specs/GeneratedTests.java"]:

template = tpl.get\_template("restassured/"+f+".j2")

write(ROOT/"api"/"restassured"/f, template.render(sol=s.solution))

# Backend

if s.solution['backend']['framework'] == 'pytest':

for f in ["pyproject.toml", "conftest.py", "tests/test\_health.py"]:

template = tpl.get\_template("pytest/"+f+".j2")

write(ROOT/"backend"/"pytest"/f, template.render(sol=s.solution))

# CI/CD

for f in ["Jenkinsfile", "azure-pipelines.yaml", "aws/buildspec.yml", "gcp/cloudbuild.yaml"]:

template = tpl.get\_template("ci/"+f+".j2")

write(ROOT/"ci"/f, template.render(sol=s.solution))

typer.echo("Scaffold complete.")

@app.command()

def generate\_tests(spec: str = "solution.yaml", stories: str = "docs/stories.md"):

# Hook for LLM/OpenAPI → tests (stub)

# Parse OpenAPI, create API tests; parse Gherkin, create UI tests.

typer.echo("Generated tests from inputs.")

@app.command()

def run\_ui(headed: bool = False):

p = ROOT/"ui"/"playwright"

subprocess.check\_call(["npm", "i"], cwd=p)

cmd = ["npx", "playwright", "test"] + (["--headed"] if headed else [])

subprocess.check\_call(cmd, cwd=p)

if \_\_name\_\_ == "\_\_main\_\_":

app()

**Agent registry & list\_agents() helper**

Add this helper to tools/agent/main.py to clearly see **which sub‑agents** are available and where they live.

# --- Agent Registry (add near top-level, below other defs) ---

AGENTS = {

"planner": "main.py:plan",

"scaffolder": "main.py:scaffold",

"req2test\_ui": "generators/story\_to\_tests.py:generate\_from\_stories",

"req2test\_api": "generators/openapi\_to\_tests.py:generate\_from\_openapi",

"ci\_writer": "main.py:scaffold (CI section)",

"runner\_ui": "main.py:run\_ui",

}

@app.command()

def list\_agents():

"""Print registered agents and their entrypoints."""

for key, val in AGENTS.items():

typer.echo(f"{key:14} -> {val}")

**Usage**

python tools/agent/main.py list-agents

Extend with adapters (Selenium, Postman/newman, Testcontainers) as plugins under tools/agent/plugins/.

**Template samples (Jinja2)**

**Playwright POM** – ui/playwright/src/pages/LoginPage.ts.j2

export class LoginPage {

constructor(private page){ }

async goto(){ await this.page.goto("{{ sol.base\_url if sol.get('base\_url') else 'https://example.com' }}"); }

async login(u:string,p:string){

await this.page.fill('#username', u);

await this.page.fill('#password', p);

await this.page.click('button[type=submit]');

}

}

**Playwright spec** – ui/playwright/tests/login.spec.ts.j2

import { test, expect } from '@playwright/test';

import { LoginPage } from '../src/pages/LoginPage';

test.describe('Smoke: Login', () => {

test('valid user can login', async ({ page }) => {

const login = new LoginPage(page);

await login.goto();

await login.login(process.env.USERNAME!, process.env.PASSWORD!);

await expect(page.getByText(/dashboard/i)).toBeVisible();

});

});

**RestAssured base** – api/restassured/src/test/java/base/ApiTest.java.j2

import io.restassured.RestAssured;

import org.junit.jupiter.api.BeforeAll;

public class ApiTest {

@BeforeAll

static void setup(){

RestAssured.baseURI = System.getenv().getOrDefault("BASE\_URL", "https://api.example.com");

}

}

**pytest backend** – backend/pytest/tests/test\_health.py.j2

def test\_db\_health(db\_conn):

assert db\_conn.ping() is True

**Data‑driven examples**

**TypeScript** (Playwright) – CSV loader

// ui/playwright/tests/utils/data.ts

import fs from 'fs';

export function csv(path:string){

return fs.readFileSync(path,'utf-8').trim().split('\n').slice(1).map(r=>{

const [username,password] = r.split(',');

return { username, password };

});

}

**Java** (TestNG DataProvider)

@DataProvider(name="users")

public Object[][] users(){

return new Object[][]{{"tom","secret"},{"jane","secret"}};

}

**Environment & secrets**

* Config files: env/config.{env}.yaml merged at runtime.
* Secrets: .env (local) and provider adapters (Vault/AWS/Azure/GCP) – only **read** at runtime; avoid committing.

env/config.qa.yaml

base\_url: https://qa.acme.com

api:

base\_url: https://qa.api.acme.com

backend:

db:

uri: postgresql://qa\_user:${DB\_PASS}@db.qa.local:5432/app

**CI/CD emitters (minimal, parameterized)**

**Jenkinsfile.j2**

pipeline {

agent any

options { timestamps() }

stages {

stage('UI') {

steps { dir('ui/playwright') { sh 'npm ci && npx playwright install --with-deps && npx playwright test' } }

post { always { archiveArtifacts allowEmptyArchive: true, artifacts: 'playwright-report/\*\*'; publishHTML([reportDir: 'playwright-report', reportName: 'Playwright Report', reportFiles: 'index.html']) } }

}

stage('API') {

steps { dir('api/restassured') { sh 'mvn -B -U test' } }

post { always { junit 'target/surefire-reports/\*.xml' } }

}

stage('Backend') {

steps { dir('backend/pytest') { sh 'pip install -r requirements.txt && pytest -q --maxfail=1 --disable-warnings --junitxml=report.xml' } }

post { always { junit 'backend/pytest/report.xml' } }

}

}

}

**Azure Pipelines** – azure-pipelines.yaml.j2

trigger: [ main ]

pool: { vmImage: 'ubuntu-latest' }

stages:

- stage: Test

jobs:

- job: UI

steps:

- script: |

cd ui/playwright

npm ci

npx playwright install --with-deps

npx playwright test

displayName: Run Playwright

- task: PublishBuildArtifacts@1

inputs: { pathToPublish: 'ui/playwright/playwright-report', artifactName: 'ui-report' }

- job: API

steps:

- script: mvn -B -U test

workingDirectory: api/restassured

- job: Backend

steps:

- script: |

pip install -r requirements.txt

pytest -q --junitxml=report.xml

workingDirectory: backend/pytest

**AWS CodeBuild** – aws/buildspec.yml.j2

version: 0.2

phases:

install:

commands:

- n 20

- npm i -g pnpm

- pip install -r backend/pytest/requirements.txt || true

build:

commands:

- cd ui/playwright && pnpm i && npx playwright install --with-deps && npx playwright test

- cd ../../api/restassured && mvn -B -U test

- cd ../../backend/pytest && pytest -q --junitxml=report.xml

artifacts:

files:

- '\*\*/\*'

**Google Cloud Build** – gcp/cloudbuild.yaml.j2

steps:

- name: gcr.io/cloud-builders/npm

dir: ui/playwright

args: ['ci']

- name: mcr.microsoft.com/playwright

dir: ui/playwright

args: ['npx','playwright','test']

- name: gcr.io/cloud-builders/mvn

dir: api/restassured

args: ['-B','test']

- name: gcr.io/cloud-builders/pip

dir: backend/pytest

args: ['install','-r','requirements.txt']

- name: python

dir: backend/pytest

args: ['-m','pytest','-q','--junitxml=report.xml']

artifacts:

objects: { location: gs://$PROJECT\_ID-artifacts/${\_REPO}/, paths: ['\*\*/\*'] }

**Reporting options**

* **Allure** unified view: add Playwright, JUnit5, and pytest adapters; publish in CI.
* **Playwright HTML report**: instant UI coverage.
* **Extent Reports** (Java) if stakeholders prefer.
* **ReportPortal** (server) for historical trends and flaky tracking.

Playwright config snippet for Allure:

// playwright.config.ts (partial)

reporter: [['list'], ['allure-playwright']],

JUnit5 + Allure (Java):

<!-- pom.xml -->

<dependency>

<groupId>io.qameta.allure</groupId><artifactId>allure-junit5</artifactId><version>2.24.0</version>

</dependency>

pytest:

# pytest.ini

addopts = --allure-dir=./allure-results

**Requirements → tests (agent logic)**

1. **Parse** stories/acceptance criteria. If Gherkin given, map directly to step templates.
2. **Classify** test type (smoke/regression/negative) + layer (UI/API/backend) using tags.
3. **Generate** skeletons:
   * UI: Page objects from sitemap/selectors found via Playwright codegen/locators.
   * API: One test per operation (positive + boundary), auth flows from security scheme in OpenAPI.
   * Backend: Health, cron, DB migrations, queues (via Testcontainers).
4. **Data‑drive** from examples tables or CSV/JSON stubs.

Start with deterministic templates; enable LLM augmentation later.

Sample LLM “story→test” system prompt (editable):

You are a senior QA automation architect. Convert the following user stories and acceptance criteria into executable test cases for {{stack}}.

- Use the POM or API spec patterns from the repository templates.

- Label each case with @smoke/@regression and @ui/@api/@backend.

- Prefer robust locators and idempotent API assertions.

Return code only.

**Requirement→Test Generation – Concrete implementation (MVP)**

This section adds **working generators** and templates so the agent can turn user stories/Gherkin/OpenAPI into executable tests.

**New files (key ones):**

tools/agent/generators/

\_\_init\_\_.py

utils.py

story\_to\_tests.py # Stories & Gherkin → Playwright tests

openapi\_to\_tests.py # OpenAPI → RestAssured tests

tools/agent/templates/

playwright/spec.spec.ts.j2

restassured/OperationTest.java.j2

docs/

stories.md # story blocks (or use docs/features/\*.feature)

specs/

api.yaml # OpenAPI 3.x

**Wire the generator (update tools/agent/main.py)**

# add to imports at top

from tools.agent.generators.story\_to\_tests import generate\_from\_stories

from tools.agent.generators.openapi\_to\_tests import generate\_from\_openapi

@app.command()

def generate\_tests(

spec: str = "solution.yaml",

stories: str = "docs/stories.md",

features: str = "docs/features",

openapi: str = "specs/api.yaml",

):

data = yaml.load(Path(ROOT / spec).read\_text())

s = Solution(\*\*data)

generate\_from\_stories(root=ROOT, sol=s.solution,

stories\_path=ROOT / stories,

features\_dir=ROOT / features)

if (ROOT / openapi).exists():

generate\_from\_openapi(root=ROOT, sol=s.solution,

openapi\_path=ROOT / openapi)

typer.echo("Generated tests from requirements (stories/Gherkin/OpenAPI).")

**tools/agent/generators/utils.py**

from pathlib import Path

from jinja2 import Environment, FileSystemLoader

import re

def jenv(base: Path):

return Environment(loader=FileSystemLoader(str(base)),

trim\_blocks=True, lstrip\_blocks=True)

def slug(s: str):

return re.sub(r"[^a-z0-9\_]+", "\_", s.lower()).strip("\_")

def write(path: Path, content: str):

path.parent.mkdir(parents=True, exist\_ok=True)

path.write\_text(content, encoding="utf-8")

**tools/agent/generators/story\_to\_tests.py**

from pathlib import Path

import re

from .utils import jenv, slug, write

GHERKIN\_RE = re.compile(r"^(Feature:|Scenario:|Given |When |Then |And )", re.I)

def parse\_stories\_md(md: str):

blocks, cur, mode = [], {"title": "", "gherkin": "", "labels": []}, None

for line in md.splitlines():

if line.startswith("### ") and "Story:" in line:

if cur["gherkin"]:

blocks.append(cur); cur = {"title": "", "gherkin": "", "labels": []}

cur["title"] = line.split("Story:", 1)[1].strip()

elif line.strip().lower().startswith("labels:"):

cur["labels"] = [t.strip().lstrip("@") for t in line.split(":",1)[1].split(",")]

elif line.strip().lower().startswith("gherkin:"):

mode = "gh"; cur["gherkin"] = ""; continue

elif line.strip().startswith("```") and mode == "gh":

mode = None; continue

elif mode == "gh":

cur["gherkin"] += line + "

"

if cur["gherkin"]:

blocks.append(cur)

return blocks

# Map a few common Gherkin steps → Playwright code

def gherkin\_to\_playwright\_steps(gherkin: str):

steps = []

for raw in [l.strip() for l in gherkin.splitlines() if GHERKIN\_RE.search(l)]:

# 1) Given I am on the login page

if re.search(r"^Given I am on the login page$", raw, flags=re.I):

steps.append("await page.goto(baseUrl + '/login');"); continue

# 2) When I login as 'user'/'pass' (capture values)

m = re.match(r"^When I login as '(.+)'/'(.+)'$", raw, flags=re.I)

if m:

u, p = m.group(1), m.group(2)

steps.append(f"await page.fill('#username', '{u}'); ")

steps.append(f"await page.fill('#password', '{p}'); ")

steps.append("await page.click('button[type=submit]');"); continue

# 3) Then I should see the dashboard

if re.search(r"^Then I should see the dashboard$", raw, flags=re.I):

steps.append("await expect(page.getByText(/dashboard/i)).toBeVisible();"); continue

return steps

def generate\_from\_stories(root: Path, sol: dict, stories\_path: Path, features\_dir: Path):

tpl = jenv(root / "tools" / "agent" / "templates")

ui\_out = root / "ui" / sol['ui']['framework'] / "tests"

# 1) Markdown stories

if stories\_path.exists():

blocks = parse\_stories\_md(stories\_path.read\_text(encoding="utf-8"))

for b in blocks:

steps = gherkin\_to\_playwright\_steps(b["gherkin"]) if sol['ui']['framework']=="playwright" else []

name = slug(b["title"]) or "story"

spec = tpl.get\_template("playwright/spec.spec.ts.j2").render(

name=name, labels=b["labels"], steps=steps,

base\_url\_var="process.env.BASE\_URL")

write(ui\_out / f"{name}.spec.ts", spec)

# 2) Raw .feature files

if features\_dir.exists():

for feat in features\_dir.glob("\*.feature"):

g = feat.read\_text(encoding="utf-8")

title = re.search(r"Feature:\s\*(.+)", g)

name = slug(title.group(1) if title else feat.stem)

steps = gherkin\_to\_playwright\_steps(g)

spec = tpl.get\_template("playwright/spec.spec.ts.j2").render(

name=name, labels=["ui","regression"], steps=steps,

base\_url\_var="process.env.BASE\_URL")

write(ui\_out / f"{name}.spec.ts", spec)

**tools/agent/generators/openapi\_to\_tests.py**

from pathlib import Path

import yaml

from .utils import jenv, write, slug

DEFAULT = {"get": 200, "post": 201, "put": 200, "patch": 200, "delete": 204}

def generate\_from\_openapi(root: Path, sol: dict, openapi\_path: Path):

spec = yaml.safe\_load(openapi\_path.read\_text(encoding="utf-8"))

tpl = jenv(root / "tools" / "agent" / "templates")

out = root / "api" / sol["api"]["framework"] / "src" / "test" / "java" / "specs"

for path, ops in (spec.get("paths") or {}).items():

for method, op in (ops or {}).items():

op\_id = op.get("operationId") or f"{method}\_{slug(path)}"

responses = op.get("responses") or {}

status = next(iter(responses.keys()), DEFAULT.get(method.lower(), 200))

java = tpl.get\_template("restassured/OperationTest.java.j2").render(

class\_name=f"{slug(op\_id).title().replace('\_','')}Test",

method=method.upper(),

path=path,

expected\_status=str(status),

)

write(out / f"{slug(op\_id).title().replace('\_','')}Test.java", java)

**Templates**

**tools/agent/templates/playwright/spec.spec.ts.j2**

import { test, expect } from '@playwright/test';

const baseUrl = {{ base\_url\_var }} || 'https://example.com';

test.describe('{{ name }}', () => {

test('{{ (labels||[]).join(' ') }} {{ name }}', async ({ page }) => {

{{ '

'.join(steps) }}

});

});

**tools/agent/templates/restassured/OperationTest.java.j2**

import io.restassured.RestAssured;

import org.junit.jupiter.api.Test;

import static io.restassured.RestAssured.\*;

import static org.hamcrest.Matchers.\*;

public class {{ class\_name }} {

@Test

void api\_call\_{{ class\_name }}() {

given()

.baseUri(System.getenv().getOrDefault("BASE\_URL", RestAssured.baseURI))

.when()

.request("{{ method }}", "{{ path }}")

.then()

.statusCode({{ expected\_status }});

}

}

**Sample docs/stories.md**

### Story: Login shows dashboard for valid user

Labels: @ui, @smoke

Gherkin:

```gherkin

Scenario: Successful login

Given I am on the login page

When I login as 'tom'/'secret'

Then I should see the dashboard

#### How to run

```bash

# 1) Scaffold once

python tools/agent/main.py scaffold

# 2) Add your inputs

# - docs/stories.md (or docs/features/\*.feature)

# - specs/api.yaml

# 3) Generate tests from requirements

python tools/agent/main.py generate-tests

# 4) Run (example: UI)

python tools/agent/main.py run-ui

This MVP uses deterministic mappings. You can plug an LLM later by adding a provider call inside story\_to\_tests.py when no mapping matches.

**Component-by-component explainer (what each part does)**

**1) solution.yaml — the single source of truth**

**What it is:** A declarative spec that tells the agent *what to build and how to run it*.

**What goes here:** Names, languages, framework choices, browsers/platforms, environments, secrets provider, CI/CD targets, reporting options, and quality knobs (retries/parallelism).

**Key knobs:**

* languages.ui|api|backend → picks TypeScript/Java/Python stacks per layer.
* ui.framework → playwright or selenium (when you add that adapter). pattern sets POM/Screenplay.
* api.framework → restassured, playwright\_api, or pytest (as you add adapters).
* environments → named envs (dev|qa|stage|prod). Values resolved from env/config.<env>.yaml + secrets provider.
* cicd.providers + cicd.default → which pipelines to emit and the default one.
* reporting.providers → e.g., allure, playwright\_html, extent, reportportal.
* test\_types + data\_strategy → tags and data loading defaults the generators can use.

**Common actions:**

* Flip Playwright→Selenium or RestAssured→Playwright API by changing fields here.
* Add/remove browsers/platforms to control Playwright grid.
* Select Jenkins vs Azure vs AWS vs GCP pipeline emission.

**2) tools/agent/main.py — the orchestrator (Supervisor agent)**

**What it is:** A Typer CLI that reads solution.yaml, calls sub‑agents, writes files, and runs tests.

**Major commands:**

* plan → prints the resolved plan (useful to validate solution.yaml).
* scaffold → writes UI/API/Backend projects and CI/CD files from templates.
* generate-tests → **Requirements→Test agents** (UI from Stories/Gherkin, API from OpenAPI).
* run-ui → installs deps and runs Playwright tests locally.
* list-agents → shows available sub‑agents and their entrypoints (added in your canvas).

**How it works:** Loads YAML → validates via Solution model → selects templates based on chosen frameworks → renders with Jinja → writes to repo.

**3) tools/agent/templates/\*\* — blueprints for code & pipelines**

**What it is:** Jinja2 templates the scaffolder renders into real files.

**What goes here:**

* UI: package.json.j2, playwright.config.ts.j2, page objects and specs.
* API: pom.xml.j2, base classes, operation tests.
* Backend: pyproject.toml.j2, conftest.py.j2, sample tests.
* CI/CD: Jenkinsfile.j2, azure-pipelines.yaml.j2, aws/buildspec.yml.j2, gcp/cloudbuild.yaml.j2.

**Customize by editing templates** to match your organization (naming, plugins, secret paths).

**4) tools/agent/generators/\*\* — Requirement→Test agents**

**What it is:** Deterministic generators that convert inputs into runnable tests.

* story\_to\_tests.py (UI): Parses docs/stories.md **or** docs/features/\*.feature; maps common Gherkin steps → Playwright code; emits ui/playwright/tests/\*.spec.ts.
* openapi\_to\_tests.py (API): Reads specs/api.yaml (OpenAPI); creates one RestAssured test per path/operation using expected status codes.
* utils.py: Jinja environment, safe writers, string helpers.

**Extend:** Add more step patterns, support for tags (@smoke, @regression) to control suites, or call an LLM when a step isn’t recognized.

**5) ui/playwright/\*\* — Web UI automation (POM by default)**

**What it is:** A Playwright TypeScript project with Page Objects and specs.

**Key files:**

* playwright.config.ts → browser matrix, retries, parallelism, reporters (Allure/HTML).
* src/pages/\*.ts → POMs. Keep locators stable (roles/testIds).
* tests/\*.spec.ts → generated & hand‑written specs.
* tests/utils/data.ts → helpers for CSV/JSON data‑driven tests.

**Env usage:** BASE\_URL, credentials, and API endpoints via env vars + env/config.<env>.yaml merge.

**6) api/restassured/\*\* — API automation**

**What it is:** A Maven/JUnit5 RestAssured project.

**Key files:**

* pom.xml with dependencies (JUnit5, RestAssured, Allure adapter if enabled).
* src/test/java/base/ApiTest.java → base setup (BASE\_URL).
* src/test/java/specs/\*Test.java → generated tests from OpenAPI (one per operation).

**Extend:** Add auth filters, schema validations, and negative tests from components/schemas.

**7) backend/pytest/\*\* — backend/integration tests**

**What it is:** A Python pytest project for DB/queue/jobs and service health checks.

**Key files:**

* conftest.py → fixtures (DB connection, message broker, clock mocks). Consider Testcontainers.
* tests/test\_health.py → example.

**Extend:** Add migration checks, cron verifications, Kafka/SQS consumers.

**8) ci/\*\* — pipeline emitters**

**What it is:** Ready YAMLs/Scripts for CI providers selected in solution.yaml.

**Behavior:** Each pipeline runs UI → API → Backend stages, publishes reports (HTML/Allure/JUnit), and archives artifacts.

**Customize:** Runners, caching, matrix builds, secret stores, test splitting.

**9) env/\*\* — environment overlays & secrets**

**What it is:** YAML files with per‑env config merged at runtime.

**Pattern:** env/config.dev.yaml, env/config.qa.yaml, etc. Secrets are NOT stored here—only references (use .env locally, Vault/AWS/Azure/GCP in CI).

**10) docs/\*\* — human‑readable requirements**

**What it is:** Input format for the generators.

* stories.md → simple Markdown with Story blocks, Labels, and a fenced gherkin section.
* features/\*.feature → raw Gherkin scenarios (optional).

**11) specs/\*\* — API specifications**

**What it is:** OpenAPI/Swagger documents that drive API test generation.

**Tip:** Keep it versioned; the generator will re‑emit tests on spec changes.

**12) data/\*\* & reports/\*\***

* data/ → CSV/JSON test data used by data‑driven suites.
* reports/ → output folder for Playwright HTML/Allure/JUnit artifacts (CI pulls from here).

**13) Quality, accessibility, and flake control**

* Retries: use carefully; fix root causes (timers, network waits). Playwright traces/videos on failure.
* Accessibility: integrate axe-core in smoke flows; fail on critical 508 issues.
* Linting: add ESLint/Prettier (UI), Checkstyle/SpotBugs (API), Flake8/Ruff (Python).

**14) How to extend the agent**

* **New UI framework**: add tools/agent/templates/selenium/\*\* and branch in scaffold().
* **New CI provider**: add template under tools/agent/templates/ci/<provider>.j2 and write from scaffold().
* **Richer req→test**: expand step maps, or add an LLM call in story\_to\_tests.py when no pattern matches.

**Execution**

* Local: python tools/agent/main.py scaffold && python tools/agent/main.py run-ui
* Docker: optional docker-compose.yml with browsers + services.
* Parallelism: Playwright shard, JUnit5 parallel, pytest -n auto with pytest-xdist.
* Retries: set in Playwright config; JUnit5/pytest retry plugins.

**Security & quality guardrails**

* Never print secrets. Use masked logs and env var providers.
* Gate merges with linters + unit tests of page objects and API clients.
* Flake control: retry only idempotent failures; capture HAR/trace/video automatically.
* Accessibility: add **axe‑core** (Playwright) to smoke pack where relevant.
* 508/ADA: optional axe scan step (fail on critical violations).

**One‑week MVP plan**

**Day 1–2**: Orchestrator CLI + YAML schema + Playwright scaffolding.  
**Day 3**: RestAssured + pytest templates.  
**Day 4**: CI emitters (Jenkins + Azure).  
**Day 5**: Allure wiring + environment overlays.  
**Day 6**: OpenAPI → API tests generator (basic).  
**Day 7**: Story → Gherkin → codegen (initial), docs + examples.

**Example spec + resulting tree**

**solution.yaml** (short):

solution:

name: demo‑shop

languages: { ui: typescript, api: java, backend: python }

ui: { framework: playwright, pattern: POM, browsers: [chromium, firefox] }

api: { framework: restassured, inputs: { openapi: specs/shop.yaml } }

backend: { framework: pytest }

test\_types: [smoke, regression]

environments: [dev, qa]

cicd: { providers: [jenkins, azure], default: jenkins }

reporting: { providers: [allure, playwright\_html] }

Result:

demo‑shop/

ui/playwright/...

api/restassured/...

backend/pytest/...

ci/Jenkinsfile

env/config.dev.yaml

env/config.qa.yaml

**Next steps**

* Tell me your **minimal initial spec** (fill solution.yaml) and a **couple of user stories** or an **OpenAPI file**.
* I’ll expand these templates for your exact stack (Selenium or Playwright, RestAssured or Playwright API, Allure vs Extent, Jenkins/Azure/AWS/GCP) and add the agents for requirement→test generation.