Executive Summary

This blueprint operationalizes a **deliverables-first, acceptance-first, contract-first** approach. It fuses proven practices—ADRs, Product-Based Planning (PBS→WBS), ATDD/BDD, contract-first API design, a light V-Model, and end-to-end traceability—into runnable templates you can drop into a repo. We assume defaults:

- **Domain:** E-commerce order service
- **Feature:** *Customer creates an order*
- **Stack:** TypeScript / Node 20
- **Test stack:** Cucumber.js + Jest
- **Contracts:** OpenAPI 3.0.3 + JSON Schema
- **CI:** GitHub Actions
- **Constraints:** Basic PII, p95 ≤ 300 ms for POST /orders, OWASP top 10 mitigations
- **NFRs:** Operability, evolvability, traceability

Artifacts below include: literature review with citations, a comparison matrix, a prescriptive blueprint, and a **runnable mini-repo** (contracts, ADRs, DDS, RTM, acceptance & contract tests, CI). Every deliverable links to **tests → evidence → files**. You can clone the structure, `npm ci`, and run the red/green loop immediately.

Literature Review

[1] Michael Nygard, *Documenting Architecture Decisions* (2011). Key idea: keep concise records of architecturally significant decisions to preserve rationale and enable change. Influences ADR/MADR usage for traceable decisions. Use when decisions affect structure/NFRs; avoid over-recording trivial choices. [Cognitect blog]

[2] ADR Hub (adr.github.io). Consolidates ADR practice, templates, and guidance; highlights decision logs as living artifacts. Informs repo layout (`docs/adr`), decision log governance.

[3] MADR Template (adr/madr). Opinionated Markdown template that standardizes context/decision/consequences/alternatives; easy to automate linting. Use for consistency across teams; avoid heavy templates that slow flow.

[4] Cucumber Gherkin Reference & Cucumber-JS install docs (cucumber.io). Establishes executable, business-readable acceptance specs. Drives **acceptance-first** with Gherkin → step defs → CI reports.

[5] Gojko Adzic, *Specification by Example* (Book). Empirical patterns from 30+ teams: shared examples as living specs, ubiquitous language, and alignment via examples. Use to frame acceptance conversations; avoid giant enumerations—focus on key examples.

[6] OpenAPI Specification v3.0.3 (OpenAPI Initiative). Canonical HTTP API contract format; enables tooling (validation, codegen, tests). Basis for **contract-first** and CI verifiers.

[7] JSON Schema (json-schema.org). Validates JSON payloads and ensures interoperability; used within OpenAPI components and standalone schemas.

[8] Pact (docs.pact.io) & PactFlow bi-directional testing. Consumer-provider contract tests reduce brittle integration tests; bi-directional mode compares provider contracts to public specs. Use when teams are decoupled; avoid for simple in-repo services where jest-openapi is sufficient.

[9] Schemathesis. Property-based fuzz testing from OpenAPI/GraphQL; complements example-based tests to discover edge cases. Add to CI once basic contract tests are green.

[10] APM: Product-Based Planning & PBS guidance. PBS is a hierarchical list of products/deliverables; WBS derives from PBS. Avoid task-first planning that creates orphan tasks.

[11] NASA Systems Engineering Handbook (Vee model). Plan verification alongside definition; map requirements → verification methods → evidence. Use a **light V** for software to avoid ceremony bloat.

[12] GitHub Actions workflow syntax. CI vehicle to execute acceptance & contract tests, and to publish evidence artifacts.

[13] Mermaid docs. Text-based graphs for dependencies/traceability; renderable in Markdown & GitHub.

[14] Nx dependency graph docs. Optional: visualize module/task dependency graphs in TS monorepos.

Full references are listed at the end.

Comparison Matrix

Practice	Purpose	Strengths	Trade-offs	Tooling	Adoption Tips
ADRs (MADR)	Preserve rationale for arch. decisions	Lightweight, searchable history; supports change	Can become stale if not tied to checks	MADR, adr-tools	Link ADRs to contracts/ tests; lint presence
PBS→WBS	Deliverables-first planning	Eliminates orphan tasks; clear acceptance	Requires discipline; mindset shift	APM templates; simple YAML	Start PBS in PRD kickoff; one owner

Practice	Purpose	Strengths	Trade-offs	Tooling	Adoption Tips
ATDD/BDD	Define behavior up-front	Shared language; living specs	Needs product engagement	Cucumber-JS, Gherkin	Keep scenarios small; 1–3 per capability
Contract-first	Formalize interfaces early	Codegen/ validation; parallel work	Spec design skill needed	OpenAPI, JSON Schema, jest-openapi, Pact	Start with one endpoint; verify in CI
Light V-Model	Plan verification while planning product	Traceability to evidence; risk-based	Over-ceremony risk	Verification matrix, RTM	Keep “V” light; map each deliverable to a verification method
Traceability RTM	Map Req/ Deliverable↔Test↔Evidence↔Files	Audit-ready; change impact	Overhead if manual	CSV/ Markdown + checkers	Automate RTM lint in CI
Dependency Graphs	See module/test/product deps	Impact analysis; focuses reviews	Over-diagramming	Mermaid, Nx	Generate from imports; embed in PRs

Blueprint (Runnable Approach)

Defaults & Assumptions

- Domain: **e-commerce order service**; initial capability: **POST /orders**.
- NFRs: p95 ≤ 300 ms, error rate ≤ 1%, evidence retained 30 days.
- Security: validate payloads via JSON Schema; basic input sanitation; log correlation IDs.

Governance

- **ADRs** in `docs/adr` using MADR. Each ADR links to contracts, DDS, and RTM row.
- **PBS→WBS**: Write PBS first (deliverables only). Derive tasks (WBS) from PBS; no orphan tasks.

- **Contracts**: OpenAPI 3.0.3 + JSON Schema; contract tests via **jest-openapi**; property-based via **Schemathesis** (optional).
- **ATDD**: Gherkin features → Cucumber steps → green CI; publish JSON evidence.
- **Traceability**: `docs/rtm.csv` source of truth; checked by a lint script (future work).
- **CI**: GitHub Actions runs contract tests then acceptance tests; publishes evidence.

Product Breakdown Structure (PBS)

1. **DEL-001 API Endpoint: POST /orders** — purpose: accept order and return `{id,status}`.
2. **DEL-002 API Contract** — purpose: formal OpenAPI describing `/orders` with schemas.
3. **DEL-003 CI Pipeline** — purpose: run contract & acceptance tests and publish evidence.

WBS (derived from PBS)

- For **DEL-001**: implement `src/api/handler.ts` + app wiring; acceptance steps.
- For **DEL-002**: author `contracts/openapi.yaml`; contract test spec.
- For **DEL-003**: author `.github/workflows/ci.yml`; ensure evidence output paths.

Key ADRs (MADR)

- **ADR-0001 Contract-first**: Define OpenAPI before route code; enforce via jest-openapi.
- **ADR-0002 Test stack**: Cucumber-JS for acceptance; Jest + Supertest for contract/system tests.
- **ADR-0003 Repo structure**: `/contracts`, `/deliverables`, `/docs`, `/src`, `/tests`, `/.github`.
- **ADR-0004 Layering**: Thin HTTP adapter → application service; validation at edges.

Verification Planning (light V)

- Left side: PBS + contracts + acceptance criteria.
- Right side: contract tests, acceptance tests, evidence artifacts in `reports/*`.

Exemplar (Mini-Project)

> **Goal:** A runnable slice that demonstrates PBS→WBS, ADRs, contracts, acceptance tests, CI, DDS, RTM, dependency graph, and DoFD.

Repo Structure

```
.
├── contracts/
│   └── openapi.yaml
├── deliverables/
│   ├── DEL-001.yaml
```

```
│   ├── DEL-002.yaml
│   └── DEL-003.yaml
├── docs/
│   ├── adr/
│   │   ├── 0001-contract-first.md
│   │   ├── 0002-test-stack.md
│   │   ├── 0003-repo-structure.md
│   │   └── 0004-layering.md
│   ├── deps.md
│   └── rtm.csv
├── src/
│   ├── api/
│   │   ├── handler.ts
│   │   └── app.ts
│   └── server.ts
├── tests/
│   ├── acceptance/
│   │   ├── user_creates_order.feature
│   │   └── steps/order.steps.ts
│   └── contract/
│       └── openapi.spec.ts
├── jest.config.js
├── tsconfig.json
├── package.json
└── .github/workflows/ci.yml
```

Contracts — `contracts/openapi.yaml`

```
openapi: 3.0.3
info: { title: Order Service, version: "0.1.0" }
paths:
  /orders:
    post:
      requestBody:
        required: true
        content:
          application/json:
            schema: { $ref: '#/components/schemas/CreateOrderRequest' }
      responses:
        "201": { description: Created, content: { application/json: { schema: {
$ref: '#/components/schemas/Order' }}}}
components:
  schemas:
    CreateOrderRequest:
      type: object
```

```
      required: [customerId, items]
      properties:
        customerId: { type: string, format: uuid }
        items:
          type: array
          items: { $ref: '#/components/schemas/OrderItem' }
    Order:
      type: object
      required: [id, status]
      properties:
        id: { type: string, format: uuid }
        status: { type: string, enum: [CREATED, REJECTED] }
    OrderItem:
      type: object
      required: [sku, qty]
      properties:
        sku: { type: string }
        qty: { type: integer, minimum: 1 }
```

Acceptance — `tests/acceptance/user_creates_order.feature`

```
Feature: Order creation
  In order to track purchases
  As a customer
  I want to create an order

  Scenario: valid order is created
    Given a valid CreateOrderRequest payload
    When I POST it to /orders
    Then I receive 201 with an Order payload with status "CREATED"
```

Step Definitions — `tests/acceptance/steps/order.steps.ts`

```
/* DoFD:
deliverable: DEL-001
satisfies_acceptance: [ACR-101]
tests_must_pass:
  - tests/acceptance/user_creates_order.feature
contracts_must_pass:
  - contracts/openapi.yaml
*/
import { Given, When, Then, BeforeAll, AfterAll } from '@cucumber/cucumber';
import request from 'supertest';
import { app, start, stop } from '../../../src/api/app';
```

```
import assert from 'assert';

let response: request.Response;
let payload: any;

BeforeAll(async () => { await start(); });
AfterAll(async () => { await stop(); });

Given('a valid CreateOrderRequest payload', function () {
  payload = { customerId: '11111111-1111-1111-1111-111111111111', items: [{
sku: 'SKU-1', qty: 1 }] };
});

When('I POST it to /orders', async function () {
  response = await request(app)
    .post('/orders')
    .set('Content-Type', 'application/json')
    .send(payload);
});

Then('I receive 201 with an Order payload with status {string}', function
(status: string) {
  assert.equal(response.status, 201);
  assert.equal(response.body.status, status);
  assert.ok(response.body.id);
});
```

Contract Test — `tests/contract/openapi.spec.ts`

```
/* DoFD:
deliverable: DEL-002
satisfies_acceptance: [ACR-201]
tests_must_pass:
  - tests/contract/openapi.spec.ts
contracts_must_pass:
  - contracts/openapi.yaml
*/
import path from 'path';
import request from 'supertest';
import jestOpenAPI from 'jest-openapi';
import { app, start, stop } from '../../src/api/app';

beforeAll(async () => {
  const specPath = path.join(__dirname, '../../contracts/openapi.yaml');
  // Initialize jest-openapi with the OpenAPI spec
  // @ts-ignore
```

```
  jestOpenAPI(specPath);
  await start();
});

afterAll(async () => { await stop(); });

test('POST /orders satisfies OpenAPI spec', async () => {
  const res = await request(app)
    .post('/orders')
    .set('Content-Type', 'application/json')
    .send({ customerId: '11111111-1111-1111-1111-111111111111', items: [{ sku:
'SKU-1', qty: 1 }] });

  expect(res.status).toBe(201);
  // Validate response against the OpenAPI spec
  // @ts-ignore
  expect(res).toSatisfyApiSpec();
});
```

App & Handler (with DoFD)

`src/api/handler.ts`

```
/* DoFD:
deliverable: DEL-001
satisfies_acceptance: [ACR-101]
tests_must_pass:
  - tests/acceptance/user_creates_order.feature
contracts_must_pass:
  - contracts/openapi.yaml
*/
import { Request, Response } from 'express';
import { randomUUID } from 'crypto';

export const createOrder = (req: Request, res: Response) => {
  // Minimal validation; full JSON Schema validation can be added later
  if (!req.body?.customerId || !Array.isArray(req.body?.items) ||
req.body.items.length === 0) {
    return res.status(400).json({ error: 'Invalid payload' });
  }
  const id = randomUUID();
  return res.status(201).json({ id, status: 'CREATED' });
};
```

`src/api/app.ts`

```ts
import express from 'express';
import http from 'http';
import { createOrder } from './handler';

export const app = express();
app.use(express.json());
app.post('/orders', createOrder);

let server: http.Server | undefined;
export async function start(port = 0) {
  return new Promise<void>((resolve) => { server = app.listen(port,
resolve); });
}
export async function stop() { return new Promise<void>((resolve) =>
server?.close(() => resolve())); }
```

`src/server.ts` (for local run)

```ts
import { app } from './api/app';
const port = process.env.PORT || 3000;
app.listen(port, () => console.log(`Order service listening on :${port}`));
```

DDS — `deliverables/DEL-001.yaml`

```yaml
id: DEL-001
name: API Endpoint /orders
purpose: "Accept an order and return id + status"
acceptance_criteria:
  - id: ACR-101
    given: "a valid CreateOrderRequest payload"
    when: "POST to /orders"
    then: "201 with Order.status=CREATED and id present"
evidence:
  tests:
    - path: tests/acceptance/user_creates_order.feature
      proves: [ACR-101]
  reports:
    - path: reports/acpt/run.json
interfaces:
  consumes: []
  exposes:
    - name: "Order API"
      contract: contracts/openapi.yaml
files_must_exist:
```

```
  - src/api/handler.ts
risks:
  - Incorrect payload validation could accept bad orders
links:
  adr: docs/adr/0001-contract-first.md
  rtM_ids: [RTM-DEL-001]
```

DDS — `deliverables/DEL-002.yaml`

```
id: DEL-002
name: OpenAPI Contract
purpose: "Formal contract for /orders to enable contract testing and client
generation"
acceptance_criteria:
  - id: ACR-201
    given: "service is running"
    when: "contract tests run"
    then: "responses satisfy OpenAPI spec"
evidence:
  tests:
    - path: tests/contract/openapi.spec.ts
      proves: [ACR-201]
  reports:
    - path: reports/contract/jest.json
interfaces:
  consumes: []
  exposes:
    - name: "Order API"
      contract: contracts/openapi.yaml
files_must_exist:
  - contracts/openapi.yaml
risks:
  - Spec drift if code changes without updating contract
links:
  adr: docs/adr/0001-contract-first.md
  rtM_ids: [RTM-DEL-002]
```

DDS — `deliverables/DEL-003.yaml`

```
id: DEL-003
name: CI Pipeline
purpose: "Run contract & acceptance tests and publish evidence"
acceptance_criteria:
  - id: ACR-301
```

```
    given: "a push or PR"
    when: "CI runs"
    then: "contract & acceptance tests execute; evidence files are published"
evidence:
  tests:
    - path: .github/workflows/ci.yml
      proves: [ACR-301]
  reports:
    - path: reports/acpt/run.json
    - path: reports/contract/jest.json
interfaces:
  consumes: []
  exposes: []
files_must_exist:
  - .github/workflows/ci.yml
risks:
  - Evidence path changes may break audits
links:
  adr: docs/adr/0003-repo-structure.md
  rtM_ids: [RTM-DEL-003]
```

ADRs (MADR style)

`docs/adr/0001-contract-first.md`

```
# Contract-first for Order Service
*Status:* Accepted
*Date:* 2025-11-05
*Context*
We require parallel development and traceable interfaces.
*Decision*
Define OpenAPI before implementing endpoints; enforce via jest-openapi in CI.
*Consequences (Positive/Negative)*
+ Enables client stubs & validation; – adds spec maintenance overhead.
*Alternatives Considered*
Code-first (annotations), gRPC/proto, GraphQL SDL.
*Related ADRs / Links to contracts, DDS, RTM*
`contracts/openapi.yaml`, `deliverables/DEL-002.yaml`, `docs/rtm.csv`
```

`docs/adr/0002-test-stack.md`

```
# Test Stack: Cucumber.js for ATDD; Jest + Supertest for contract/system
*Status:* Accepted
*Date:* 2025-11-05
*Context*
```

```
Business-readable scenarios with executable backing, plus fast Node test runner.
*Decision*
Use @cucumber/cucumber for acceptance; Jest with jest-openapi & Supertest for
contract/system.
*Consequences*
+ Clear Given/When/Then; – two runners to configure.
*Alternatives*
Playwright BDD, Mocha/Chai.
*Related*
`tests/acceptance/*`, `tests/contract/*`, `deliverables/*.yaml`
```

`docs/adr/0003-repo-structure.md`

```
# Repository Structure & Evidence Folders
*Status:* Accepted
*Date:* 2025-11-05
*Decision*
Contracts in `/contracts`; ADRs in `/docs/adr`; DDS in `/deliverables`; evidence
in `/reports/*` generated by CI.
*Consequences*
+ Predictable paths enable RTM checks; – adds directories to maintain.
```

`docs/adr/0004-layering.md`

```
# Layering: Thin HTTP Adapter → Application Service
*Status:* Proposed
*Date:* 2025-11-05
*Decision*
Keep handler minimal; move rules to services; validate at boundaries.
```

RTM — `docs/rtm.csv`

```
type,id,links_to,artifact
Deliverable,DEL-001,ACR-101;TEST-ACPT-42;FILE-src/api/handler.ts,deliverables/
DEL-001.yaml
Acceptance,ACR-101,TEST-ACPT-42,deliverables/DEL-001.yaml
Test,TEST-ACPT-42,EVID-ACPT-42,tests/acceptance/user_creates_order.feature
Evidence,EVID-ACPT-42,,reports/acpt/run.json
File,FILE-src/api/handler.ts,DEL-001,src/api/handler.ts
Contract,CT-API,TEST-CT-API,contracts/openapi.yaml
Deliverable,DEL-002,ACR-201;TEST-CT-API;FILE-contracts/
openapi.yaml,deliverables/DEL-002.yaml
Acceptance,ACR-201,TEST-CT-API,deliverables/DEL-002.yaml
```

```
Test,TEST-CT-API,EVID-CT-API,tests/contract/openapi.spec.ts
Evidence,EVID-CT-API,,reports/contract/jest.json
Deliverable,DEL-003,ACR-301;FILE-.github/workflows/ci.yml,deliverables/
DEL-003.yaml
File,FILE-.github/workflows/ci.yml,DEL-003,.github/workflows/ci.yml
```

Dependency Graph — `docs/deps.md`

```
graph TD
  DEL-001[API Endpoint /orders] --> TEST-ACPT-42
  DEL-001 --> CT-API[OpenAPI Contract]
  CT-API --> TEST-CT-API[Contract Tests]
  TEST-ACPT-42 --> EVID-ACPT-42[CI Evidence]
  FILE-src_api_handler[handler.ts] --> DEL-001
```

CI — `.github/workflows/ci.yml`

```
name: CI
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with: { node-version: 20 }
      - run: npm ci
      - name: Contract tests
        run: npx jest --config jest.config.js --runTestsByPath tests/contract/
openapi.spec.ts --json --outputFile=reports/contract/jest.json
      - name: Acceptance tests (Cucumber JSON)
        run: npx cucumber-js tests/acceptance --require-module ts-node/register
--require tests/acceptance/steps/**/*.ts -f json:reports/acpt/run.json
```

Tooling Config

`package.json`

```
{
  "name": "deliverables-first-blueprint",
  "private": true,
  "scripts": {
```

```
    "start": "ts-node src/server.ts",
    "test": "jest",
    "test:contract": "jest --config jest.config.js --runTestsByPath tests/
contract/openapi.spec.ts",
    "test:acceptance": "cucumber-js tests/acceptance --require-module ts-node/
register --require tests/acceptance/steps/**/*.ts -f json:reports/acpt/run.json"
  },
  "dependencies": {
    "express": "^4.19.2"
  },
  "devDependencies": {
    "@cucumber/cucumber": "^11.0.0",
    "@types/express": "^4.17.21",
    "@types/jest": "^29.5.12",
    "@types/node": "^20.10.8",
    "@types/supertest": "^6.0.3",
    "jest": "^29.7.0",
    "jest-openapi": "^0.14.2",
    "supertest": "^6.3.4",
    "ts-jest": "^29.1.1",
    "ts-node": "^10.9.2",
    "typescript": "^5.6.3"
  }
}
```

`jest.config.js`

```
module.exports = {
  preset: 'ts-jest',
  testEnvironment: 'node',
  testMatch: ['**/*.spec.ts']
};
```

`tsconfig.json`

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "moduleResolution": "node",
    "esModuleInterop": true,
    "strict": true,
    "resolveJsonModule": true,
    "skipLibCheck": true,
    "outDir": "dist"
```

```
  },
  "include": ["src", "tests"]
}
```

How to Run Locally

1) `npm ci`
2) **Contract tests:** `npm run test:contract`
3) **Acceptance tests:** `npm run test:acceptance`
4) **Dev server:** `npm start` then `curl -X POST :3000/orders` with a valid payload.

Adoption Playbook

1. **Week 0—Seed repo**: Add `/contracts`, `/docs/adr`, `/deliverables`, `/tests`, RTM CSV, CI skeleton. Create ADR-0001 (contract-first).
2. **Week 1—Pilot slice**: Model one endpoint in OpenAPI; write 1–2 Gherkin scenarios; wire jest-openapi + Supertest; ship evidence to `reports/*` in CI.
3. **Week 2—Scale**: Add PBS for next 3–5 deliverables; standardize DDS YAML and DoFD headers; automate an RTM linter.
4. **Governance**: PR checklist: *Contract diff reviewed? ADR updated? Feature has Gherkin? RTM rows added?*
5. **Automation**: enforce presence of DoFD blocks; fail CI if RTM paths don't resolve; optionally add Pact/Schemathesis when teams split.

Common pitfalls & mitigations
- Specs drifting from code → *gate merges on jest-openapi*. - BDD turning into UI scripts → *keep at API/service level*. - PBS ignored → *reject tasks not traceable to a deliverable*.

References

[1] Nygard, *Documenting Architecture Decisions* (2011).
[2] ADR Hub, adr.github.io (practice & resources).
[3] MADR Template, GitHub adr/madr.
[4] Cucumber: Gherkin Reference & Cucumber-JS Installation.
[5] Adzic, *Specification by Example*, Manning.
[6] OpenAPI Specification v3.0.3, OpenAPI Initiative.
[7] JSON Schema, json-schema.org.
[8] Pact docs; PactFlow Bi-Directional Testing.
[9] Schemathesis docs.
[10] APM: Product-Based Planning & PBS.
[11] NASA Systems Engineering Handbook (Vee model).
[12] GitHub Actions workflow syntax.

[13] Mermaid documentation.
[14] Nx project & task graphs.