THE SELF-CODING STACK · FREE COURSE

Build software that codes, heals & ships itself.

The AI-First Framework — the foundations, free. The mini-SaaS that builds itself, the self-coding loop, and the pipeline-vs-supervised-loop shift. Read it anywhere.

self-coding-stack-i8fky.ondigitalocean.app

The Mini-SaaS That Builds Itself

> **The big idea:** you're not bolting a chatbot onto a SaaS. You're building **one app** where AI is a teammate that lives *inside* the product — and that teammate ships features, hunts bugs, tunes performance, and patches security holes **without leaving the app**.
>
> This lesson is the spine of everything else. By the end you'll have the full mental model for an **AI-first mini-SaaS**: small enough to build in a weekend, real enough to run in production.

Why "mini-SaaS"?

Most tutorials teach you to *call* an AI API. That gives you a feature. We're after something bigger and more useful: a **small product that maintains itself**.

A mini-SaaS is the perfect size to learn this on:

- Small enough that you can hold the whole thing in your head.
- Real enough to have users, data, money, bugs, and an attack surface.
- Complete enough that all four "self-coding" jobs show up: **build, debug, optimize, secure.**

You'll build one product. The same AI core does four jobs inside it. That's the whole trick — **one app, one AI control plane, four superpowers.**

The four jobs (all in one app, all with AI)

This is the architecture in one picture. Everything in the course is a deeper cut of one of these four loops.

Job	What the AI does inside your app	The loop
🛠️ **Develop**	Proposes and writes new features as plugins, behind a review gate	spec → draft → self-review → ship
🐞 **Debug**	Detects failures, finds root cause, writes the patch	signal → diagnose → fix → verify
⚡ **Optimize**	Watches its own metrics, finds the slow/expensive path, tunes it	measure → spot → tune → re-measure

Job	What the AI does inside your app	The loop
🔒 **Secure**	Reviews every change and input for risk, blocks or flags it	inspect → judge → gate → log

The magic isn't any single one — it's that they run on the **same foundation**. Build that foundation once and all four become possible.

The foundation: five systems

Underneath the four jobs are five systems. Think of them as the body of your product.

1. **The Nervous System — See.** Structured events, metrics, and state the AI can read. If the AI can't see it, it can't act on it. Observability isn't a dashboard here — it's the AI's senses.
2. **The Brain — Control.** The AI as a control plane that acts only through **typed tools** (not free-form shell). Every action is a function with a schema and a permission.
3. **The Immune System — Heal & Fix.** Watches the nervous system for trouble, recovers, and ships patches for root causes.
4. **The Conscience — Review.** An *independent* AI pass that critiques every change before it lands. The thing that proposes a change is never the thing that approves it.
5. **The Growth Engine — Expand.** New capabilities arrive as plugins the system can propose, write, review, and load.

> **Why typed tools, not "let the AI run commands"?** Because a tool is a contract. It has a name, a schema, and a blast radius you decide. The trust ladder (below) is only possible because every action is a tool you can rank by risk.

The trust ladder (how you sleep at night)

AI-first does **not** mean "AI does whatever it wants." It means autonomy is **earned, bounded, and reversible**. Every tool sits on a rung:

```
Rung 0  Read-only        →  see, summarize, suggest           (auto, always)
Rung 1  Reversible write →  retry, cache, flag, open a draft   (auto)
Rung 2  Bounded change   →  patch behind a review gate + tests (auto + gate)
Rung 3  Risky / costly   →  schema change, spend, deploy       (human approves)
Rung 4  Irreversible     →  delete data, rotate secrets        (human + 2nd factor)
```

You add autonomy by **promoting a tool up a rung** once you trust it — not by flipping a master switch. This is the single most important idea for shipping AI-first safely.

How the four jobs use the five systems

Here's the satisfying part — none of the four jobs needs new infrastructure. They're all the five systems pointed at a different goal:

- **Develop** = Brain proposes a plugin → Conscience reviews it → Growth Engine loads it.
- **Debug** = Nervous System flags an error → Immune System diagnoses → patch through the Conscience gate.
- **Optimize** = Nervous System exposes latency/cost → Brain finds the hot path → tune behind the gate.
- **Secure** = Conscience inspects every change *and* every risky input → gates or escalates per the trust ladder.

One foundation. Four jobs. That's an AI-first product.

What you'll actually build in this course

A runnable mini-SaaS (TypeScript scaffold provided) that:

- Emits structured events the AI can read (See)
- Exposes a small set of typed tools the AI acts through (Control)
- Detects a seeded failure and recovers + patches it (Heal & Fix)
- Runs an independent review gate on every AI change (Review)
- Loads a new feature as a plugin the AI proposed (Expand)
- Refuses unsafe actions via the trust ladder (Secure)

Start tiny: wire **See + Control** and let the AI *suggest* (Rung 0). Then earn each rung. By the capstone you'll have a product that genuinely participates in building, fixing, tuning, and defending itself — and the judgment to know which rung each new power belongs on.

Try it now (2-minute exercise)

Take any app you've built. Write one sentence for each job:

1. **Develop:** "A new feature I'd let AI *draft* (not merge) is _____."
2. **Debug:** "The one error my app could detect and retry on its own is _____."
3. **Optimize:** "The slow/expensive path AI should watch is _____."
4. **Secure:** "The one input I'd never let through without a review is _____."

Those four sentences are your mini-SaaS's first four tools. The rest of the course turns them into a system. Next lesson: the Nervous System — giving your product senses the AI can read.

The Self-Coding Loop (SCL) — The Framework

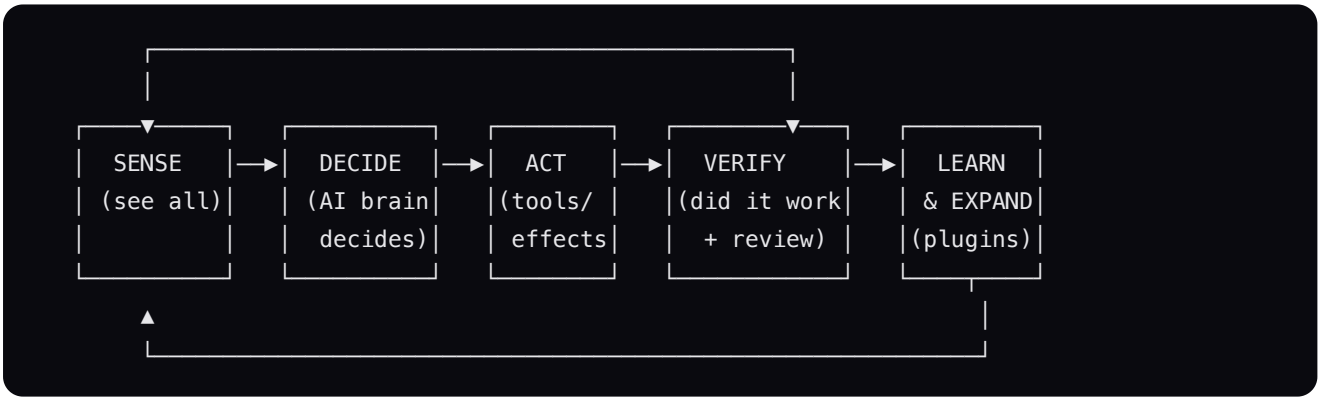
> The backbone of the entire course. Every tier teaches you to build the same thing — a product that codes, watches, heals, reviews, and expands itself — using this framework. It is distilled from the **AI Skeleton SaaS** architecture and made stack-agnostic so you can apply it to any product.

---

1. The core idea

A normal app is a **pipeline**: request → code you wrote → response. A human is the only thing that observes, decides, fixes, and extends it.

An **AI-first system** is a **loop**. The system continuously:



The human supervises the loop instead of being a step inside it. **That single shift — from pipeline to supervised loop — is what "AI-first" actually means.**

---

2. The 6 pillars → 5 systems

The six capabilities (See · Control · Heal · Fix · Review · Expand) are delivered by **five architectural systems**, using a biological metaphor so they're easy to reason about:

#	System	Biological analogy	Pillar(s)	Loop stage
1	**Nervous System**	senses + signals	**See**	Sense
2	**Brain**	decision-making	**Control**	Decide / Act
3	**Immune System**	detect + repair	**Heal**, **Fix**	Verify (recover)
4	**Conscience**	self-critique	**Review**	Verify (gate)
5	**Growth Engine**	growing new tissue	**Expand**	Learn / Expand

Build them in this order. Each one is useless without the one before it — **you cannot heal what you cannot see, and you cannot let AI act until it can review itself.**

---

3. The five systems in detail

System 1 — The Nervous System (SEE)

Goal: the system can observe its entire self in a form AI can reason over.

- **Event log** — every meaningful state change is an append-only event.
- **Unified state model** — schemas designed for machine reasoning, not just storage.
- **Telemetry** — structured logs, traces, metrics emitted for consumption by AI, not humans.
- **The Context Layer** — the function that turns raw state into the *right slice* of context for a given decision (this is the most important and most-skipped piece).

> **Rule of the Nervous System:** if the AI can't see it, it can't act on it. Observability is not ops hygiene here — it's the substrate of autonomy.

System 2 — The Brain (CONTROL)

Goal: AI is the control plane that decides and acts across the whole system.

- **The agent loop:** perceive (pull context) → decide (LLM reasons) → act (call tools) → verify.
- **Tools = the system's hands.** Every action the system can take is a typed tool the AI may call.
- **Memory:** short-term (this task), long-term (vector store), episodic (what happened before).
- **The Orchestrator:** routes work to specialized agents; chooses the model tier (cost/quality).

- **Trust boundaries:** a policy layer declaring what the AI may do autonomously vs. with approval.

> **Rule of the Brain:** the AI never touches the world directly — only through typed tools with declared blast radius. Tools are where safety lives.

System 3 — The Immune System (HEAL + FIX)

Goal: the system detects failure and recovers/repairs without a human.

- **Detection:** health signals + anomaly detection on the telemetry from System 1.
- **Heal (fast path):** AI-controlled retries, fallbacks, circuit-breakers — restore service *now*.
- **Fix (slow path):** AI reads logs/traces → root-cause → generates a patch → tests it in a sandbox → proposes (or, within trust boundaries, applies) it.
- **Incident memory:** every incident becomes an episodic memory so the system gets wiser.

> **Rule of the Immune System:** heal first (stop the bleeding), fix second (cure the cause). Never let auto-fix run without the Conscience (System 4) gating it.

System 4 — The Conscience (REVIEW)

Goal: AI critiques AI before anything lands.

- **The self-review gate:** any AI-generated change (a fix, a new plugin) is reviewed by a *separate* AI pass before merge.
- **Review panels:** multiple reviewers with distinct lenses — correctness, security, performance — vote; a majority/critical-fail blocks the change.
- **Human-in-the-loop:** anything outside the trust boundary escalates to you with the AI's reasoning attached.

> **Rule of the Conscience:** the generator and the reviewer must be independent passes. Self-review by the same call that wrote the code is theater.

System 5 — The Growth Engine (EXPAND)

Goal: the product proposes and builds new features as plugins.

- **Plugin architecture:** features are hot-swappable units with a declared contract (inputs, outputs, permissions). This is what makes self-expansion *safe*.
- **Ideation:** AI mines user data + telemetry (System 1) for feature opportunities.
- **Generation:** AI writes the plugin — spec → code → tests — using the Brain (System 2).
- **Gating:** the new plugin passes the Conscience (System 4) before registration.
- **Registration:** the plugin is loaded; the loop now has a new capability.

> **Rule of the Growth Engine:** expansion only exists if it composes the other four systems. A product "codes itself" when generation (Brain) + review (Conscience) + observability (Nervous System) close the loop.

---

4. The trust ladder (how much autonomy to grant)

Don't go fully autonomous on day one. Graduate the system up this ladder per capability:

Level	The AI…	Use when
L0 — **Observe**	only sees & reports	always start here
L1 — **Suggest**	proposes actions, human executes	building confidence
L2 — **Act-with-approval**	drafts the action, human clicks approve	medium blast radius
L3 — **Act-and-notify**	acts autonomously, tells you after	low blast radius, well-tested
L4 — **Fully autonomous**	acts silently within policy	proven, reversible, bounded

Every tool/capability has its own level. Auto-heal might be L4 (reversible retry) while auto-fix-and-deploy stays L2 for a long time. **Blast radius decides the level, not vibes.**

---

5. The build order (what you'll actually do)

1. **Nervous System** — event log + state model + telemetry + context layer. *(See)*
2. **Brain** — agent loop + tools + memory + orchestrator + trust policy. *(Control)*
3. **Immune System** — detection + auto-heal, then auto-fix in a sandbox. *(Heal/Fix)*
4. **Conscience** — self-review gate + review panels. *(Review)*
5. **Growth Engine** — plugin contract + AI generation + gated registration. *(Expand)*
6. **Close the loop** — wire Sense→Decide→Act→Verify→Learn so it runs continuously.

The course tiers map onto this exactly:

- 🟢 **Foundations (low):** Systems 1–2 (See + Control) — the loop's first half.
- 🔵 **Cohort (medium):** Systems 3–5 (Heal/Fix + Review + Expand) + close the loop + capstone.
- 🟣 **Mastermind (high):** apply all five to *your* product, with the real AI Skeleton SaaS source patterns.

---

6. The starter skill

The `skill/create-ai-first-project` scaffolds a working project pre-wired with all five systems as stubs (and a runnable loop), so you implement the framework instead of plumbing. Every tier uses it; higher tiers fill in more of it. See that folder's `SKILL.md`.

---

7. The one-paragraph summary (memorize this)

> An AI-first product is a **supervised loop**, not a pipeline. Build a **Nervous System** so the AI can see everything, a **Brain** that decides and acts only through typed tools with declared blast radius, an **Immune System** that heals then fixes, a **Conscience** that independently reviews every AI change before it lands, and a **Growth Engine** that turns reviewed AI-written plugins into new capability. Grant autonomy per-capability up a **trust ladder** governed by blast radius. Do that, and the product codes itself while you supervise.

Module 0 — Orientation: What "AI-First" Actually Means

Framework: Systems 1–5 overview (this module sets up the whole loop before you build Systems 1–2). Pillars: See · Control · Heal · Fix · Review · Expand.

This module rewires how you think about architecture before you write a line of code. If you only take one thing from it, take this: **an AI-first product is a supervised loop, not a pipeline.** Everything else in the course is a consequence of that sentence.

Learning objectives

By the end of this module you will be able to:

- Explain the difference between a **pipeline** (AI as a button) and a **supervised loop** (AI as the control plane), and recognize which one any given "AI app" really is.
- Name the **6 pillars** (See · Control · Heal · Fix · Review · Expand) and the **5 systems** that deliver them, in build order, and say why the order is non-negotiable.
- Describe how a production AI-first system like **AI Skeleton SaaS** heals and fixes itself.
- Read the **trust ladder** (L0–L4) and reason about which level a capability belongs at using *blast radius*.
- Run the scaffold's `npm run demo` and **trace the loop** end to end (Lab 0).

The core concept: pipeline vs. supervised loop

A normal application is a **pipeline**. Data flows one way through code a human wrote:

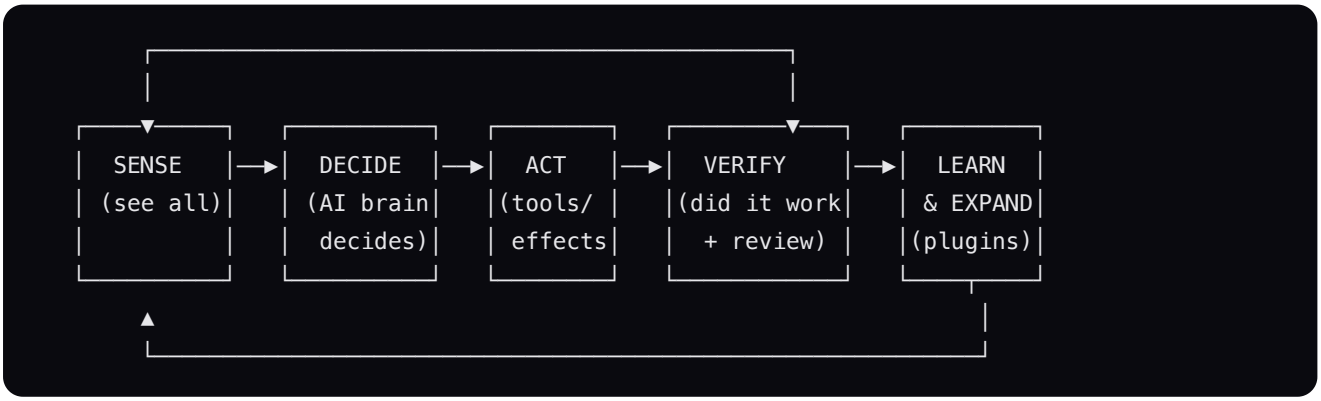
```
request ⟶ your handler ⟶ your business logic ⟶ database ⟶ response
```

The only thing that *observes* this pipeline, *decides* it's misbehaving, *fixes* it, and *extends* it is a human — you, on call, at 3am. The software cannot reason about itself. When people "add AI" to a pipeline, they usually staple an LLM call into one box:

```
request ⟶ handler ⟶ [ LLM call ] ⟶ response
```

This is the **bolt-on trap**. The model is a *button*: the user (or your code) presses it, it returns text, and it goes back to sleep. It can't see the rest of the system, it can't take actions, and you can't trust it with anything that matters — so it stays a toy.

An **AI-first system** is a **loop**. The system continuously runs:



The shift is small to state and enormous in consequence: **the human supervises the loop instead of being a step inside it.** You're no longer the observer/decider/fixer in the hot path; the

system does that, and you watch it, set its boundaries, and raise its autonomy as it earns trust. *That single move — pipeline → supervised loop — is what "AI-first" means.*

Why this matters for you as a builder

- **It scales past your attention.** A pipeline's reliability is capped by how many dashboards a human can watch. A loop's reliability is capped by how good your *signals* and *boundaries* are — which you can keep improving.
- **It makes the AI useful, not decorative.** A model that can see system state and act through tools is a control plane. A model that returns a paragraph is a chatbot.
- **It forces good architecture.** You literally cannot build the loop without observability, typed actions, and explicit trust. The framework makes you do the right things in the right order.

The 6 pillars → 5 systems

The six capabilities an AI-first system needs are **See · Control · Heal · Fix · Review · Expand**. They're delivered by **five architectural systems**, named with a biological metaphor so they're easy to reason about:

#	System	Biological analogy	Pillar(s)	Loop stage	Tier
1	**Nervous System**	senses + signals	**See**	Sense	🟢 Foundations
2	**Brain**	decision-making	**Control**	Decide / Act	🟢 Foundations
3	**Immune System**	detect + repair	**Heal**, **Fix**	Verify (recover)	🔵 Cohort
4	**Conscience**	self-critique	**Review**	Verify (gate)	🔵 Cohort
5	**Growth Engine**	growing new tissue	**Expand**	Learn / Expand	🔵 Cohort

Build them in this order. Each is useless without the one before it. You cannot heal what you cannot see (System 3 needs System 1's telemetry). You cannot let the AI act autonomously until something can review its actions (System 2's autonomy is gated by System 4). The order isn't a suggestion; it's a dependency graph.

This tier — **Foundations** — builds Systems **1 and 2**: the part of the loop that *sees* and *decides/acts*. The rest comes in later tiers.

Quick tour of each system (so you know where you're headed)

- **System 1 — Nervous System (SEE):** an append-only event log, a state model designed for *machine* reasoning, structured telemetry, and the **Context Layer** that turns all of that

into the right slice of context for a given decision. *Rule: if the AI can't see it, it can't act on it.* (Module 1.)

- **System 2 — Brain (CONTROL):** the agent loop (perceive → decide → act → verify), a registry of **typed tools** the AI calls, memory, an orchestrator that routes work and picks the model tier, and **trust boundaries**. *Rule: the AI never touches the world directly — only through typed tools with declared blast radius.* (Module 2.)
- **System 3 — Immune System (HEAL + FIX):** detect anomalies in telemetry, *heal* fast (retry/fallback/restart), then *fix* the cause (AI generates a patch, tests it in a sandbox, proposes it). *Rule: heal first, fix second.* (Cohort.)
- **System 4 — Conscience (REVIEW):** every AI-generated change is reviewed by a *separate* AI pass before it lands — review panels with distinct lenses (correctness, security, performance) vote. *Rule: the generator and the reviewer must be independent passes.* (Cohort.)
- **System 5 — Growth Engine (EXPAND):** features are hot-swappable **plugins** with a declared contract; the AI ideates from telemetry, writes the plugin, the Conscience gates it, and it gets registered. *Rule: expansion only exists if it composes the other four.* (Cohort.)

The AI Skeleton SaaS example: a system that heals and fixes itself

AI Skeleton SaaS is the production AI-first platform this course uses as its reference architecture. Here's a concrete, representative example of its loop in motion — the kind of behavior you are working toward.

The incident. At 02:14, AI Skeleton SaaS's `lesson-render` service starts throwing 500s because a downstream embedding provider degrades and request latency climbs. In a pipeline app, a human gets paged.

In AI Skeleton SaaS, the loop handles it:

1. **SENSE.** The Nervous System has been emitting structured events the whole time: `request.completed`, `request.failed`, `provider.latency`, and a rolling `errorRate` metric. The error rate crosses its threshold. *Nothing here is for a human dashboard — it's emitted as machine-readable telemetry the rest of the loop consumes.*
2. **DECIDE.** The Immune System detects the anomaly and the Brain pulls a context slice: recent failures, the implicated service, the latency spike, and the *episodic memory* of the last three times this happened. It decides to **heal** first.
3. **ACT (heal).** The Brain calls the `restart_service` tool (blast radius: **low**, reversible) and trips a circuit breaker that fails embedding calls over to a cached fallback. Error rate drops. The bleeding stops in ~20 seconds. This action is **L4 autonomous** at AI Skeleton SaaS because it's reversible and well-tested — no human needed.

4. **VERIFY.** The loop re-reads telemetry. Error rate is back under threshold but the *root cause* (the degraded provider under load) is still there. Healing held the line; it didn't cure anything.
5. **FIX (slow path).** The Brain root-causes from logs/traces: the service has no backpressure, so it stampedes the failing provider. It **generates a patch** (add a connection-pool limit + backpressure), runs it in an **ephemeral sandbox** against the test suite — and the patch passes.
6. **REVIEW.** The patch does **not** ship yet. A separate **Conscience** pass reviews it through three lenses (correctness, security, performance). They approve.
7. **TRUST GATE.** Applying a code patch to production is **high** blast radius. At AI Skeleton SaaS this capability sits at **L2 — act-with-approval**, so the loop drafts the change, opens a PR with the full reasoning attached, and pings the on-call engineer to click approve in the morning. *Heal was autonomous; fix-and-deploy was not — because blast radius decides.*

When the engineer wakes up, the outage is already over (healed at 02:14), the root cause is diagnosed, the fix is written and tested, and there's a reviewed PR waiting. They read the reasoning and approve. **The human supervised the loop; they were never a step inside it.**

That's the target. In Foundations you build steps 1–3 (sense, decide, act through a gated tool). Steps 4–7 are the Cohort tier — but you should be able to *see* how the whole thing hangs together now.

The trust ladder (intro)

You do **not** go fully autonomous on day one. Autonomy is granted **per capability**, and you graduate each one up this ladder:

Level	The AI…	Use when
L0 — Observe	only sees & reports	always start here
L1 — Suggest	proposes actions, human executes	building confidence
L2 — Act-with-approval	drafts the action, human clicks approve	medium blast radius
L3 — Act-and-notify	acts autonomously, tells you after	low blast radius, well-tested
L4 — Fully autonomous	acts silently within policy	proven, reversible, bounded

The deciding factor is **blast radius**, not how confident the model sounds. In the AI Skeleton SaaS story above, `restart_service` (reversible, low radius) ran at **L4**, while `apply_patch` to production (high radius) stayed at **L2**. Same system, same incident, different autonomy — because the *cost of being wrong* differs.

In the scaffold this is real code. `src/trust.ts` defines the levels and a single function that gates execution:

```
// src/trust.ts (excerpt)
const ALLOWED: Record<TrustLevel, BlastRadius[]> = {
  L0_observe:           ["none"],
  L1_suggest:           ["none"],
  L2_act_with_approval: ["none", "low"],
  L3_act_and_notify:    ["none", "low", "medium"],
  L4_autonomous:        ["none", "low", "medium", "high"],
};

export function canActAutonomously(level: TrustLevel, radius: BlastRadius): boolean {
  return ALLOWED[level].includes(radius);
}
```

The project's default is **L1 — Suggest**. Foundations lives at L0–L1: the AI sees and proposes; you stay in the act seat. You'll earn higher rungs in later tiers, capability by capability. *Blast radius decides the level, not vibes.*

Lab 0 — Run the loop and trace it

Goal: get the scaffold running and read the loop's own trace so the five systems stop being abstract.

Setup

Use the **`create-ai-first-project`** skill to scaffold your project (it copies the `template/` for you). Then:

```
cd my-self-coding-product
npm install
npm run demo
```

The demo runs offline on the deterministic stub LLM — **no API key needed**.

What the demo does

Read `src/demo.ts` alongside the output. It:

1. Runs **Tick 1** on a healthy system.
2. **Injects a failure**: `loop.ns.recordMetric("errorRate", 0.8)` and emits an `alert` event into the Nervous System.

3. Runs **Tick 2** on the now-failing system — watch it detect, heal, and (if heal doesn't hold) propose a fix that goes through the Conscience before any patch is applied.
4. Runs **`expand()`** — the Growth Engine proposes a plugin, gated by the Conscience.
5. Prints the **loop trace**: every `sense`, `decide`, `act`, `verify`, `heal`, `fix.*`, and `expand` step.

Your task

1. Run `npm run demo` and `npm test`. Both should pass.
2. In the loop trace, **find and label one line for each loop stage**: SENSE, DECIDE, ACT, VERIFY, (HEAL), and EXPAND. Write a one-sentence note next to each saying what happened.
3. Open `src/loop.ts` and match each trace line to the code in `tick()` that produced it.
4. In `src/trust.ts`, find why the demo's actions execute or defer. The default trust is `L1_suggest`. **Predict:** at L1, which of the three tools in `src/tools.ts` (`log_observation` = none, `restart_service` = low, `apply_patch` = high) can the Brain run autonomously? Then verify against the trace.
5. **Experiment:** set `SCL_TRUST=L3_act_and_notify` and re-run (`SCL_TRUST=L3_act_and_notify npm run demo`). What changed about which actions executed vs. deferred? Explain it in terms of blast radius.

Acceptance criteria

- [ ] `npm run demo` runs and prints a loop trace; `npm test` passes.
- [ ] You can point to one trace line per loop stage and explain it.
- [ ] You correctly predicted which tools run autonomously at L1 (only `none`-radius: `log_observation`) and confirmed it.
- [ ] Changing `SCL_TRUST` visibly changes which actions execute, and you can explain why using blast radius.

You don't write production code in Lab 0 — you build the *mental model*. Modules 1 and 2 are where you replace the stubs with real systems.

How this maps to the framework + trust ladder

- This module is the **whole loop** at a glance; the rest of Foundations zooms into Systems **1 (Nervous System / SEE)** and **2 (Brain / CONTROL)** — the **Sense** and **Decide/Act** stages.
- You met the **trust ladder** and lived on its bottom rungs: the demo defaults to **L1 — Suggest**, and you observed how blast radius gates execution. Foundations stays L0–L1

by design; you graduate capabilities later.
- The one-paragraph summary to memorize: *An AI-first product is a supervised loop, not a pipeline. Build a Nervous System so the AI can see everything, a Brain that decides and acts only through typed tools with declared blast radius, an Immune System that heals then fixes, a Conscience that independently reviews every AI change before it lands, and a Growth Engine that turns reviewed AI-written plugins into new capability. Grant autonomy per-capability up a trust ladder governed by blast radius.*

Knowledge check

1. In one sentence, what is the difference between a pipeline and a supervised loop, and where does the human sit in each?
2. List the five systems in build order and give the one-line reason the order can't change.
3. In the AI Skeleton SaaS incident, `restart_service` ran autonomously (L4) but `apply_patch` to production waited for approval (L2). What property decided the difference, and why?
4. The framework says "if the AI can't see it, it can't act on it." Which system does that rule belong to, and which later system depends on it most directly?
5. At trust level **L1 — Suggest**, which blast radii can execute without human approval? What does the AI do with everything else?

Common mistakes

- **Calling a pipeline "AI-first" because it has an LLM call.** If the model is a button in one box and can't see or act on the system, it's bolt-on, not AI-first.
- **Skipping the Nervous System to get to the "cool" AI part.** You can't decide or heal without seeing. Build in order or you'll be debugging blind.
- **Setting autonomy by confidence instead of blast radius.** "The model is usually right" is not a reason to let it deploy to prod. Reversibility and blast radius decide the rung.
- **Treating the trust ladder as a global setting.** It's *per capability*. Heal can be L4 while fix-and-deploy is L2 in the same system at the same time.
- **Reading the demo output without reading `loop.ts` and `trust.ts`.** The trace only teaches you something if you map each line back to the code that produced it.

Next: **Module 1 — The Data Foundation** — build the Nervous System so the AI can actually see.