Chapter Six

Plant Overview

The first screen that is allowed to say a number came from the plant

Everything so far has been preparation, and preparation is easy to mistake for progress. Five chapters produced a workspace that compiles, a palette that resolves, thirty-three routes that navigate, four signals that update, and a client that can ask a question — and not one of them has yet been asked to work with any of the others. This chapter takes the shortest possible path through all five and finds out. When it closes, one entry on the map turns from ▒ `not yet` to ■ `built`, and it is the only entry in the book that has to prove the architecture rather than merely use it.

Plant Overview is the right screen to prove it with, for a reason that has nothing to do with it being first in the sidebar. It is the screen an operator leaves open. It aggregates four subsystems into a single glance, and it is the one place in the console where being wrong is not a display bug but a safety event — because an operator who trusts this screen will not go looking at the others. That makes it the hardest screen in the book to port faithfully, and porting it faithfully requires refusing to draw parts of what `index.html` currently draws.

WHERE WE ARE ON THE MAP

```
NEXUS-1 — Angular Console        Ch. 6 · Plant Overview

Overview ............................ ■ THIS CHAPTER
Plant Fleet ......................... ~ stubbed
Plant 3D View ....................... ~ stubbed
Training Mode ....................... ~ stubbed
Reactor  (8 screens) ................ ~ stubbed
Rod Inspection ...................... ~ stubbed
Personnel ........................... ~ stubbed
Plant Lifecycle ..................... ~ stubbed
Robotics & Vehicles ................. ~ stubbed
Zone Access ......................... ~ stubbed
Power & Grid · Radiation / Safety ... ~ stubbed
Alarms · AI · Optimization · Deps ... ~ stubbed
Registry · Twin · Trends · Root Cause ~ stubbed
Audit · Security · NX-Script · Help . ~ stubbed

▸ THIS CHAPTER — the first ■ built. Every other entry is reachable
  and explicit about being empty. This one stops being a placeholder
  and starts making claims about a reactor.

Routes registered: 33 of 33 · Screens built: 1 of 33 · Specs green: 27
```

SEE IT IN NEXUS-1

Open `index.html` and find `<section class="page active" data-page="overview">`. It is three grids: a `.grid.c4` of four `.stat` cards; a `.grid.c-21` holding the `.mimic` energy-flow strip and a six-row `Subsystem Status` panel; and a `.grid.c-12` holding the output chart and the alarm list. Read the markup carefully before reading the rest of this chapter — specifically, look at how many of the values in it are `<span class="live" data-sig="…">` placeholders filled by the simulation, and how many are simply typed into the document. The count is the subject of this chapter.

WHAT THE SCREEN CLAIMS

Before porting a screen it is worth writing down what it asserts, because a component is easier to review against a list of claims than against a picture. Plant Overview makes eleven, and they are not equally well founded.

Four are **measurements**: electrical output in MWe, thermal power in MWt, reactor power as a percentage of rated flux, and the five values along the energy-flow strip. In the source file these are driven by `data-sig` attributes and updated by the simulation, which means they are at least *modelled* — the file computes them from a coherent physical chain rather than picking them at random. Six are **subsystem verdicts**: Reactor Core nominal, Primary Coolant nominal, Turbine/Generator on watch, Radiation Monitors normal, Safety Systems armed, Grid Connection synced. And one is **availability**, printed as 99.4 %.

The last seven are typed into the markup. Not computed, not simulated, not derived — literals, sitting in the document since the day it was written, and rendered with exactly the same visual authority as the four that are modelled. The `99.4` in the availability card is a string in an HTML file. So is `Watch · vib`, the amber pill telling an operator that a turbine has a vibration problem.

HONEST BOUNDARY — SEVEN CLAIMS THIS CHAPTER REFUSES TO PORT AS-IS

The six subsystem verdicts and the availability figure have no source. Volume III's API exposes nothing that answers "is the turbine on watch," because that verdict is a judgement about a machine, not a reading from one, and the trilogy never modelled it. Porting them as literals would carry a fabrication across a rewrite whose entire justification was that the console must stop inventing numbers.

So this chapter ports them as *absent*. The subsystem panel renders each row with a `NO SOURCE` pill in `--text-mute`, and the availability card renders its value slot empty with the same marker. Both are visually quiet and unmistakably not verdicts. **Chapter 19** wires the six rows to real subsystem health once Power & Grid and Radiation/Safety are built; **Chapter 26** computes availability from historical uptime, which is the only place the number can legitimately come from. Until then the screen says nothing rather than something convenient.

This is the decision the whole book turns on and it is worth being blunt about how it feels: the ported screen looks *worse* than the original. Six confident green pills become six grey ones. A crisp 99.4% becomes a blank. Any reasonable stakeholder seeing both side by side will prefer the file we are replacing, and the answer to that preference is that the original screen was more reassuring because it was less truthful.

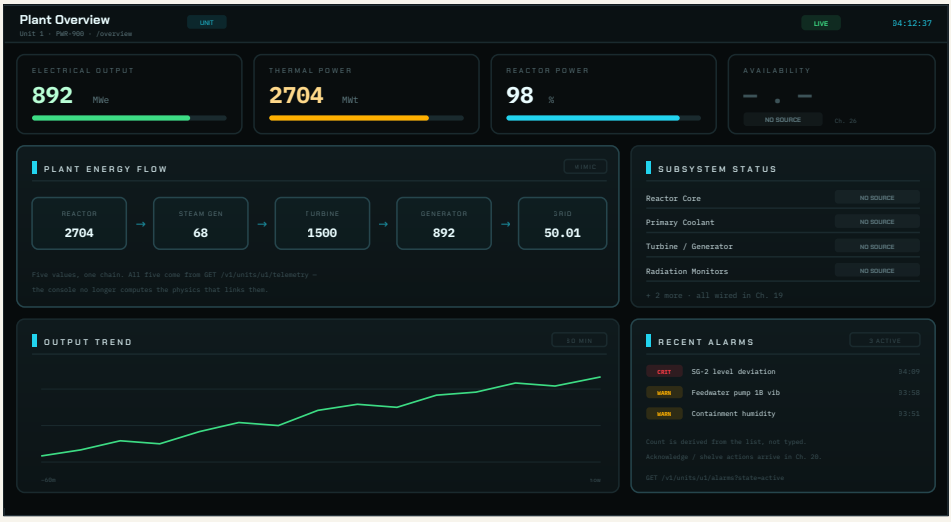

Figure 6.1 — Plant Overview as this chapter builds it. Compare it against Figure 1.1 and count the differences: three cards carry live values and the fourth admits it has no source; the energy-flow strip is unchanged in appearance but every number in it now arrives over HTTP; the subsystem panel has traded six confident verdicts for six declared blanks; and the alarm count in the corner is derived from the list beside it rather than typed into the markup. This is what the rest of the book looks like: the same console, making fewer and better-founded claims.

THE COMPONENT

The class is smaller than the screen suggests, because everything it displays it reads and nothing it displays it computes. Three signals arrive from the API service; a fourth is derived; and the two panels with no backing data are handled by rendering a marker rather than by any logic at all.

THE OVERVIEW COMPONENT, IN FULL — FEATURES/OVERVIEW/OVERVIEW.TS

```ts
import { Component, computed, inject, signal } from '@angular/core';
import { toSignal } from '@angular/core/rxjs-interop';
import { PlantApi } from '../../core/api/plant-api';
import { PlantStateService } from '../../core/state/plant-state';

@Component({
  selector: 'nx-overview',
  templateUrl: './overview.html',
  styleUrl: './overview.scss',
})
export class OverviewComponent {
  private readonly api   = inject(PlantApi);
  private readonly state = inject(PlantStateService);

  // The unit whose numbers this screen shows. Read, never stored.
  readonly unit = this.state.selected;

  // Telemetry for that unit. Re-fetched whenever the selection changes.
  readonly telemetry = toSignal(
    this.api.telemetryFor(this.state.selectedId),
    { initialValue: null },
  );

  readonly alarms = toSignal(
    this.api.activeAlarmsFor(this.state.selectedId),
    { initialValue: [] },
  );

  // Derived, not typed. The card and the list can never disagree.
  readonly alarmCount = computed(() => this.alarms().length);

  // The four subsystems Ch. 19 will wire. Declared here so the panel
  // renders its rows now, each explicitly sourceless.
  readonly subsystems = signal([
    { name: 'Reactor Core',        verdict: null },
    { name: 'Primary Coolant',     verdict: null },
    { name: 'Turbine / Generator', verdict: null },
    { name: 'Radiation Monitors',  verdict: null },
    { name: 'Safety Systems',      verdict: null },
    { name: 'Grid Connection',     verdict: null },
  ]);
}
```

Two details are worth pausing on. `alarmCount` is a `computed` over the same list the panel renders, which is the rule from Chapter 4 applied to a number an operator

reads: the badge saying *3 active* and the three rows beneath it cannot drift, because one is derived from the other. In `index.html` the count is the string `3 active` and the rows are pushed by a separate function, so they can and eventually do disagree.

And `subsystems` holds `verdict: null` six times. That is not a placeholder awaiting a value — it is the value. `null` here means *no source exists for this claim*, and the template renders it as such. When Chapter 19 supplies real health data the shape does not change; only the nulls do.

> **FRAMEWORK NOTE — `toSignal` AND `computed`, IF THEY ARE NEW**
>
> Two pieces of machinery from Chapter 4 do all the work above, and neither needs its Deep Dive re-read. `toSignal(source, {initialValue})` takes something that produces values over time — here an HTTP poll — and gives back a signal you read in a template as `telemetry()`. The `initialValue` is what it holds before the first response lands, which is why `null` appears there and why the template needs an `@else` branch.
>
> `computed(() => …)` declares a value derived from other signals. It re-evaluates by itself whenever anything it read has changed, and never at any other time. You do not call it to refresh it and you cannot assign to it. That is the whole reason `alarmCount` is written this way: a value you cannot set is a value that cannot disagree with the list it was computed from.

THE TEMPLATE — ENERGY FLOW AND SUBSYSTEM STATUS

FEATURES/OVERVIEW/OVERVIEW.HTML · 2 OF 3 GRIDS, FULL FILE IN APPENDIX E

```
<section class="grid c-21">

  <div class="panel hot">
    <div class="ph"><h2><span class="tick"></span>Plant Energy Flow</h2>
      <span class="tag">Mimic</span></div>

    @if (telemetry(); as t) {
      <div class="mimic">
        <div class="mnode"><div class="mt">Reactor</div>
          <div class="mv">{{ t.thermalMwt | number:'1.0-0' }} MWt</div></div>
        <div class="marrow">&rarr;</div>
        <div class="mnode"><div class="mt">Steam Gen</div>
          <div class="mv">{{ t.steamBar | number:'1.0-0' }} bar</div></div>
        <div class="marrow">&rarr;</div>
        <div class="mnode"><div class="mt">Turbine</div>
          <div class="mv">{{ t.turbineRpm | number:'1.0-0' }}</div></div>
        <div class="marrow">&rarr;</div>
        <div class="mnode"><div class="mt">Generator</div>
          <div class="mv">{{ t.electricalMwe | number:'1.0-0' }} MW</div></div>
        <div class="marrow">&rarr;</div>
        <div class="mnode"><div class="mt">Grid</div>
          <div class="mv">{{ t.gridHz | number:'1.2-2' }} Hz</div></div>
      </div>
    } @else {
      <div class="mimic offline"><span class="nosource">NO TELEMETRY</span></div>
    }
  </div>

  <div class="panel">
    <div class="ph"><h2><span class="tick"></span>Subsystem Status</h2></div>
    @for (s of subsystems(); track s.name) {
      <div class="row"><div class="ln">{{ s.name }}</div>
        @if (s.verdict) {
          <span class="pill" [class]="s.verdict.level">{{ s.verdict.text }}</span>
        } @else {
          <span class="pill nosource">NO SOURCE</span>
        }
      </div>
    }
  </div>
</section>
```

The `@else` branch on the mimic is the two-state rule arriving on a real screen for the first time. If telemetry has not loaded — or the interceptor from Chapter 5 classified the attempt as offline — the strip does not render five nodes with dashes in them. It renders one marker saying there is no telemetry, because five empty boxes in the shape of a working plant is a more misleading picture than a declared absence.

THE SPECS THAT HOLD THE LINE

Three of this chapter's eight specs are ordinary — the screen renders, the values appear, the units are right. The five worth printing are the ones that fail if a later chapter quietly re-introduces a fabrication.

THE SPECS THAT ENFORCE HONESTY — FEATURES/OVERVIEW/OVERVIEW.SPEC.TS

```
it('renders NO SOURCE for every unsourced subsystem', () => {
  const pills = el.queryAll(By.css('.row .pill'));
  expect(pills.length).toBe(6);
  for (const p of pills) {
    expect(p.nativeElement.textContent).toContain('NO SOURCE');
    expect(p.nativeElement.classList).toContain('nosource');
  }
});

// The regression guard. If someone re-adds a literal verdict, this fails.
it('never renders an ok/warn/crit pill without a verdict object', () => {
  const rows = el.queryAll(By.css('.row'));
  rows.forEach((row, i) => {
    const hasVerdict = component.subsystems()[i].verdict !== null;
    const signalPill = row.query(By.css('.pill.ok, .pill.warn, .pill.crit'));
    expect(!!signalPill).toBe(hasVerdict);
  });
});

it('derives the alarm badge from the list, not a literal', () => {
  api.emitAlarms([alarmA, alarmB]);
  fixture.detectChanges();
  expect(tag('.ph .tag').textContent).toContain('2 active');
  expect(el.queryAll(By.css('.alarm-row')).length).toBe(2);
});

it('shows NO TELEMETRY rather than empty nodes when offline', () => {
  api.failTelemetry({ status: 0 });
  fixture.detectChanges();
  expect(el.queryAll(By.css('.mnode')).length).toBe(0);
  expect(tag('.nosource').textContent).toContain('NO TELEMETRY');
});

it('shows no availability figure at all', () => {
  const card = tag('[data-card="availability"] .v');
  expect(card.textContent).not.toMatch(/\d/);
});
```

The second spec is the one to steal. It does not assert what the screen shows; it asserts that what the screen shows *agrees with whether a source exists*. A future contributor who hard-codes a green pill to make a demo look better fails a test whose name explains exactly why. That is a more durable guarantee than a code-review convention, and it costs nine lines.

The last one is deliberately strange. It asserts the absence of any digit in the availability card — a test that would be absurd on most screens a[illegible] is the point on this one

It will be deleted in Chapter 26, and deleting it will be the moment the number becomes legitimate.

HONEST BOUNDARY — WHAT THIS SCREEN STILL CANNOT DO

The trend chart is rendered, not interactive. It draws sixty minutes of electrical output as a polyline over the same SVG viewBox `index.html` used. There is no hover readout, no zoom, and no range selection, and the charting decision itself is deferred to **Chapter 26**, where it is made once and applied to every later chart rather than guessed at here.

Alarms are read-only. The list renders and the badge counts, but acknowledging or shelving an alarm is a `POST` against a real endpoint with real consequences, and it belongs with the rest of the alarm workflow in **Chapter 20**.

Polling is naive. Telemetry is re-fetched on an interval; there is no backoff when the backend is refusing, and no WebSocket. **Chapter 11** replaces this with a real-time transport when the kinetics screens make the difference visible.

CHECKPOINT

1. Eleven claims were identified on this screen. How many are backed by an API response, and what happens visually to the other seven?
2. Why is `alarmCount` a `computed` rather than a number the API returns alongside the list? What class of bug does that choice make impossible?
3. The mimic strip renders *nothing* when telemetry is unavailable, rather than five nodes containing dashes. Argue the case for the other choice, then say why this book rejects it.
4. Which spec would fail first if a contributor re-added `<span class="pill ok">Nominal</span>` to the subsystem panel, and what would its failure message tell them?
5. The availability spec asserts that a card contains no digits. Name the chapter that deletes it and explain what has to be true before it can be deleted.

Map state carried into Ch. 7 — Routes: **33 of 33** · Screens built: **1 of 33** · Specs green: **27**. Overview is the first ■ `built`. Chapter 7 makes the unit selector real and watches this screen react.

Provenance: Where Every Pixel's Number Came From, and How to Tell

THE CHAPTER'S STORY IN ONE PANEL

```
features/overview/  overview.ts · overview.html · overview.scss
                    overview.spec.ts — 8 specs, 5 of them about honesty
Claims on screen:   11 total
  4 measured        telemetry endpoint · Volume III
  1 derived         alarm count, computed from the list
  6 unsourced       rendered as NO SOURCE — Ch. 19 and Ch. 26
Screens built: 1 of 33 · first ■ on the map
```

The five previous Dives taught framework machinery, because a reader who did not know what a signal was could not follow the chapter. From here that job is done. This Dive and the ones after it teach the harder discipline the rest of the book runs on: auditing what a screen asserts, and knowing which assertions it has earned.

Every value on a screen has a provenance

Pick any number visible in a console and ask one question: *what would have to be true for this to be wrong?* The answers sort into four classes, and the classes matter more than they first appear because a screen gives all four the same visual authority unless it is deliberately built not to.

A **measured** value came from an instrument, through the backend, over HTTP. It can be wrong if the instrument drifts or the request is stale, and those failures are detectable — a timestamp tells you the second one. A **derived** value is computed from measured ones by a rule you can read: the alarm count, a percentage of rated power. It is wrong only if its inputs are wrong or the rule is, and both are inspectable. A **modelled** value came from a simulation — every number in `index.html` is this class — and can be entirely self-consistent while bearing no relationship to any physical plant. A **fabricated** value was typed into the source by a person and has no relationship to anything at all; it cannot become wrong because it was never right.

The rewrite's whole purpose is to move values from the third and fourth classes into the first and second. Chapter 6's audit found seven fabrications on the single most-watched screen in the console, and none of them were malicious — they were placeholders that stopped looking like placeholders the moment the file started being demonstrated.

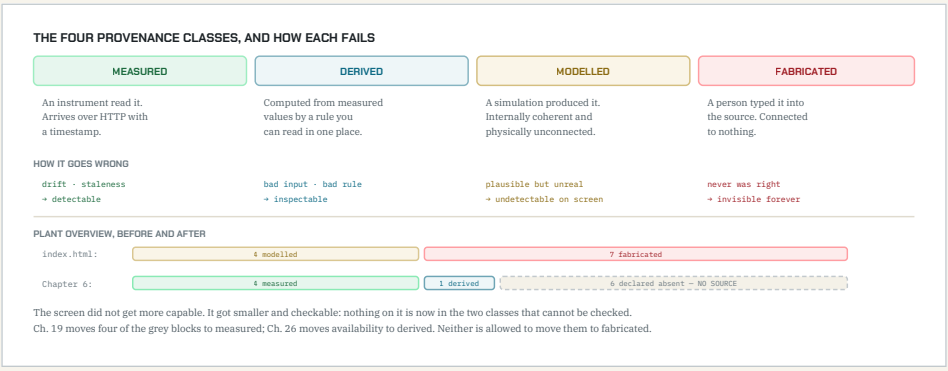

Figure D6.1 — The audit that opens every remaining chapter of this book. The two right-hand classes share a property that makes them dangerous on an operator console: nothing visible on the screen distinguishes them from the two on the left. A fabricated 99.4 % is rendered in the same typeface, at the same size, in the same card as a measured 892 MWe.

Why absence has to be loud, but not alarming

Having decided not to show a fabricated value, a screen still has to render something, and the choice is more delicate than it looks. Showing **nothing at all** — omitting the card — is tempting and wrong, because a missing panel is indistinguishable from a panel that was never designed. An operator who has used this console for a year knows availability sits in the fourth card; removing it silently teaches them the console has changed rather than that a source is missing. Showing a **zero or a dash in the normal style** is worse: it reads as a measurement of zero, which on an availability card is a plant-wide emergency.

The third option is what Chapter 6 does. The card keeps its position and label, its value slot renders in `--text-mute` rather than the bright value colour, and a small pill states the reason. It is unmistakably not a reading, and equally unmistakably not an alarm — the console's red is spent on actual plant conditions and must not be diluted by the application's own incompleteness. This is why `.nosource` is styled from `--text-mute` and not from `--amber`.

Colour is a claim too

The audit is easy to run on numbers and easy to forget on everything else, which is why the definition of “claim” in this book is deliberately wide. A pill's colour asserts a severity. A bar's fill fraction asserts a proportion of some maximum. A chart's vertical scale asserts a range worth caring about. Each of these can be fabricated exactly as a number can, and each is harder to spot because none of them looks like data.

The `.stat` cards make the point. Each has a `data-max` attribute — 1100 for MWe, 3300 for MWt, 110 for flux — and the bar underneath fills to the measured value as a fraction of it. Those maxima are the plant's rated capacities, so the bar is a real claim about headroom. But `110` for reactor power is someone's decision that 110 % of rated flux is the top of the scale, and an operator reading a nine-tenths full bar is reading that decision as much as the measurement. Chapter 6 keeps all three maxima, recorded as constants with a comment naming their origin.

> **A NOTE ON DEMOS**
>
> The practical objection to all of this is that a screen full of grey blocks demonstrates badly, and the objection is real. The answer is not to fill them in for the demo. It is to show the map panel beside the screen: six `NO SOURCE` markers next to a roadmap naming Chapters 19 and 26 reads as a plan, whereas six green pills that later turn out to be markup reads as something else entirely, usually at the worst possible moment.

The audit, as a procedure

Every remaining chapter opens with the same four steps, and they take about twenty minutes per screen. First, **enumerate the claims** — walk the source markup and list every value, label, and status a user would read as a statement about the plant. Second, **classify each** against the four boxes in Figure D6.1; the fabrications are usually obvious once you are looking, because they are the ones with no `data-sig`. Third, **find the source or name the chapter** — for each claim not yet measured, either locate the endpoint in the Volume III contract or write down which future chapter supplies it. Fourth, **write the spec that fails if it comes back**, which is the step that makes the other three durable.

That fourth step is what separates this from a code review. A review catches a fabrication once; a spec catches it every time anyone runs the suite, including the contributor who re-introduces it in eighteen months with entirely good intentions because a stakeholder asked why the box was grey.

The vocabulary, once

`provenance`	Where a displayed value came from. The four classes: measured, derived, modelled, fabricated.
`claim`	Anything on screen a user reads as a statement about the plant — including a pill, a colour, and a count.
`NO SOURCE`	This book's marker for a claim with no backing data. Muted, never amber or red.
`two-state rule`	Any screen with more than one visual state shows both, and a spec forces the difference.

Chapter 7 takes the same screen and asks a different question: what happens to all of it when the operator selects a different unit? The provenance audit does not change — but every measured value on the screen has to re-fetch, and the ones that do not are a new class of lie.