— CHAPTER 14 · NEXT.JS 16 + FHIR · COLD OPEN

Last chapter we built the BFF. Now we stand in front of it and render FHIR.

Next.js sixteen, rendering real Patients, Encounters, and Observations from the NestJS BFF we built last chapter. Three goals: the one rule that governs every healthcare frontend, fetching with Server Components, and rendering FHIR resources without crashing on missing data.

What you will be able to do

- The BFF rule — the browser never touches HAPI
- Server Components — async fetch via the BFF
- Rendering resources — defensively, every time

— THE RULE · THE LOAD-BEARING POSTURE OF THIS WHOLE CHAPTER

THE BFF RULE — the browser never talks to HAPI. Ever.

HOP 1 · CLIENT

Browser

Holds no FHIR token, no HAPI URL, no clinic header. Calls only same-origin /api/v1/*.

HOP 2 · SAME-ORIGIN

Next.js route

/api/v1/* in ddx-web. Forwards the request server-side to the BFF.

HOP 3 · SERVER ONLY

ddx-api BFF

The BFF we built last chapter. Holds the bearer token and X-Clinic-ID; only it reaches HAPI :18080.

DOWNSTREAM

HAPI :18080

FHIR server. Reached by the BFF alone, with bearer + tenant header injected server-side.

— FORBIDDEN · THE SHORTCUT THAT LEAKS CREDENTIALS

~~**Browser ----> HAPI :18080**~~

Skip the BFF and the access token lives in the browser where anyone can read it, and the tenant header becomes client-controlled, breaking isolation. Three hops, token off the browser, tenant injected server-side — that is the whole posture.

ALLOWED — Browser to /api/v1/* to BFF to HAPI

FORBIDDEN — Browser direct to HAPI

— WALKTHROUGH · APP ROUTER · THE PAGE IS AN ASYNC FUNCTION

A Server Component awaits the BFF — credentials never enter the bundle.

In the App Router a page becomes an async function and awaits its data right there. Because Server Components render only on the server, the bearer and tenant header are safe to send — the query logic and credentials never reach the client bundle.

— APP/PATIENT/[ID]/PAGE.TSX · SERVER COMPONENT

export default async function Page({ params }) {

const [patient, encounters, observations] = await Promise.all([api.getPatient(params.id), api.getEncounters(params.id), api.getObservations(params.id)]); return <Chart patient={patient} encounters={encounters} observations={observations} />; }

— V16 GOTCHA · FETCH

Not cached, blocks render

In Next.js sixteen fetch is no longer cached by default and blocks render. Wrap the fetching component in Suspense to stream, or reach for the cache directive.

— V16 GOTCHA · PARALLEL

Promise.all, not waterfall

Patient, Encounters, and Observations fire together and await with Promise.all — never one after another. Or give each its own Suspense boundary to stream independently.

— FRAMEWORK ANCHOR · THE CLIENT-COMPONENT PATH (CONCEPTUAL)

On the client, a typed client calls methods — not URLs.

— CONCEPTUAL SHAPE · NOT A LITERAL SIGNATURE

const patient: Patient = await api.getPatient(id); // targets /api/v1/*

Interactive widgets and forms call a method and get back a typed FHIR resource — instead of hand-writing fetch to /api/v1/*. Server Components call the BFF directly; this typed client is the client-component path.

Ava

That is where a typed client earns its keep — one method per resource, a typed Patient back.

Marcus · one seam, not many

Exactly. Cardinal twenty — extend the existing seam. One client, one place to evolve.

— WALKTHROUGH · PATIENT · ALMOST EVERY FIELD IS OPTIONAL

Compute the display name. Show the MRN — never the resource id.

Name is an array of HumanName: pick the official use if present, else the first; render its text, or join given and family. Birth date can be year-only and still valid. Telecom and address are arrays you filter and tolerate empty.

NAME · HUMANNAME[]

Display name

use=official else first; name.text, else given + family. Never assume name[0] exists.

IDENTIFIER · IDENTIFIER[]

MRN

The identifier whose type code is MRN — the human-facing medical record number.

BIRTHDATE · DATE

Partial dates valid

A year-only birthDate is legal. Render what is there; do not pad or guess.

TELECOM · ADDRESS

Filter, tolerate empty

ContactPoint[] and Address[] — filter by system phone/email; an empty array is normal.

— THE TRAP · RESOURCE ID IS NOT THE PATIENT NUMBER

patient.id // server-assigned logical id — NOT the MRN

The resource id is the server-assigned logical id. Never surface it to a clinician as the patient number. The human-facing medical record number lives in identifier, the one with the MRN type code — show that.

— DIAGRAM · ENCOUNTER · A TIMELINE SORTED BY PERIOD.START

Sort by period.start. Badge status and class. Absent end means ongoing.

Encounters become a timeline ordered by period.start — and you handle a null start, because a planned encounter may not have one yet. References carry a display string, so render subject.display and participant.display and skip the extra fetch.

PERIOD.START · EARLIEST

2026-01-12 · AMB

Status finished. Class ambulatory. subject.display rendered straight from the reference.

finished

PERIOD.START · RECENT

2026-04-03 · IMP

Status in-progress. Class inpatient. period.end absent — so we render ongoing.

in-progress · ongoing

PERIOD.START · NULL

planned · EMER

No start yet. A planned encounter still has a place — do not drop it from the list.

planned

class AMB / IMP / EMER — ambulatory, inpatient, emergency

absent period.end renders "ongoing", never a blank cell

— WALKTHROUGH · OBSERVATION · WHERE CH 03 AND CH 04 PAY OFF

Switch on value[x]. Show the unit; compare on the UCUM code.

value[x] is a choice type. Most vitals and labs use valueQuantity — the Quantity we met in Chapter three — but it might be a CodeableConcept or a string, so switch on which one is present.

— VALUEQUANTITY · UNIT VS CODE

{ "value": 70, "unit": "kg", "system": "http://unitsofmeasure.org", "code": "kg" }

Show value and unit to the human — 70 kg. But the canonical unit is the UCUM code from Chapter four. Display unit; compare and convert on code.

— COMPONENT[] · BLOOD PRESSURE

component: [systolic, diastolic]

Blood pressure is two component entries — systolic and diastolic — each its own Quantity. Render both, each with its own value and unit.

DISPLAY LADDER · 1

.text

The human-readable original. Prefer it — callback to the CodeableConcept of Chapter three.

DISPLAY LADDER · 2

coding[].display

First coding's display, optionally the system you prefer — SNOMED or LOINC from Chapter four.

DISPLAY LADDER · 3

bare code = defect

A raw code shown to a clinician, e.g. 38341003, is a defect. Last resort only, flagged.

— CONCEPT · THE DEFINING TRAIT OF REAL FHIR DATA IS SPARSENESS

Assume absence, not presence — one bad resource degrades one card.

Nearly every element is optional, arrays come back empty, and a choice type means exactly one variant exists. The renderer must assume absence, not presence.

— BAD · ASSUMES PRESENCE, CRASHES

patient.name[0].text · coding[0].code · period.start

Index zero into an empty array and the whole chart throws. The clinician sees a blank page, not a missing field. One sparse resource takes down everything.

— GOOD · NULLISH-SAFE, DEGRADES GRACEFULLY

patient.name?.[0]?.text ?? "Not recorded"

Reach every value through nullish-safe accessors. Show a neutral placeholder — a dash, or "Not recorded" — never the word undefined.

RULE 1

Never index zero

Never assume coding[0], name[0], or period.start exist. Reach them safely.

RULE 2

Neutral placeholder

Missing renders "—" or "Not recorded" — readable, not a stack trace.

RULE 3

dataAbsentReason

Distinguish plain absence from dataAbsentReason — FHIR telling you it is missing on purpose.

RULE 4

Per-card boundary

Wrap each card in an error boundary — one malformed resource degrades one card, not the page.

— DIAGRAM · THE SYNTHESIS · ONE PAGE, THE WHOLE FLOW

Page fires Promise.all to the BFF; three components render defensively.

The page is a Server Component. It fires Promise.all to the BFF — Patient, Encounters, and Observations together. Each result flows into its own component, each rendering defensively.

— SERVER SIDE · THE LINE YOU CANNOT CROSS

Page (Server Component) ----> Promise.all ----> ddx-api BFF ----> HAPI :18080

The bearer token and the tenant header live only on the server side of this diagram. The browser never sees them.

● RENDER · SERVER

PatientCard

Display name, MRN, telecom — nullish-safe, neutral placeholders.

● RENDER · SERVER

EncounterTimeline

Sorted by period.start; status and class badges; "ongoing" for absent end.

● RENDER · SERVER

ObservationList

Switch on value[x]; unit for humans, UCUM code for compare; display ladder.

● CLIENT · INTERACTIVE

One widget

A single interactive widget takes the ClientApiClient path back to /api/v1/*.

— RECAP · TRY IT YOURSELF · AND THE BRIDGE TO CH 15

Three things to keep: the rule, the parallel fetch, the defensive render.

- The BFF rule — browser to your BFF, never to HAPI
- Server-render — fetch in parallel, credentials off the client
- Defensive — almost nothing is required

— TRY IT · ONE PROMISE.ALL, THREE RESOURCES

const [patient, encounters, observations] = await Promise.all([api.getPatient(id), api.getEncounters(id), api.getObservations(id)]);

One Promise.all to /api/v1/*, three resources at once — never hapi.fhir.org directly. That single call is the whole chapter in practice.

- Next · Ch 15 · SSE — live updates pushed to the UI when a clinician saves a new Observation