— CHAPTER 06 · ACT II OPENS — FROM THE SPEC TO A RUNNING SERVER

For five chapters we learned a language. Now we run a server that speaks it.

Resources, datatypes, terminology, Bundles — that was the FHIR spec. From here the course turns. HAPI FHIR is the reference Java implementation, the one most production systems are built on. Three goals: the modules HAPI is made of, how it stores resources in a real database, and the pattern that keeps our own server upgradeable.

Where this chapter goes

- Modules — the Maven module stack
- Storage — Hibernate and the HFJ tables
- Overlay — the upgradeable deployment pattern

— DIAGRAM · WHAT HAPI IS, UNDERNEATH

HAPI is a stack of composable Maven modules.

Read it bottom-up. Each layer depends on the one below it, and an application caps the stack by assembling them into a server.

● LAYER 1 · CORE

hapi-fhir-base

Version-independent core — parsers, the client, the FHIR context.

● LAYER 2 · MODEL

structures-r4

The R4 resource model — the classes for Patient, Observation, Condition.

● LAYER 3 · STORAGE API

hapi-fhir-storage

Storage abstractions — the DAO and persistence interfaces.

● LAYER 4 · PERSISTENCE

jpaserver-base

The concrete JPA implementation that talks to a database via Hibernate.

— VALIDATION + ASSEMBLY CAP

FhirInstanceValidator · RestfulServer + IResourceProvider beans

A validation module supplies the instance validator. To become a running server you assemble these modules — a RestfulServer with your resource-provider beans on top. That is the whole shape.

— DIAGRAM · WHERE THE RESOURCE ACTUALLY LANDS

The transaction Bundle you posted becomes rows, through Hibernate.

— CURRENT STATE

HFJ_RESOURCE

Holds the current state of every resource — one row per live resource, the version you read on a plain GET.

— EVERY VERSION

HFJ_RES_VER

Holds every historical version. The serialized body lives in the version row — that is what powers _history reads and versioned reads.

Ava

Into a relational database, through Hibernate. The resource you POST becomes rows.

Marcus · the JPA server

Hibernate ORM maps the entities onto whatever engine you run — Postgres, H2, SQL Server, Oracle.

— DIAGRAM · HOW SEARCH AVOIDS SCANNING EVERY BODY

Search parameters are extracted at write time into typed index tables.

When you save a resource, HAPI writes typed index rows. A REST search compiles straight to SQL against those tables — never a scan of the serialized bodies.

STRINGS

HFJ_SPIDX_STRING

name, address, free-text params.

TOKENS

HFJ_SPIDX_TOKEN

coded values — system plus code.

DATE · QUANTITY · URI

HFJ_SPIDX_DATE / _QUANTITY / _URI

typed columns for range and unit search.

REFERENCES

HFJ_RES_LINK

resolved links between resources for chaining.

— FULL-TEXT IS A SEPARATE ENGINE

Hibernate Search — Lucene (single-node) or Elasticsearch (scale-out)

Finding a word inside a note — _content, _text, :contains — is powered by Hibernate Search. Lucene on disk is the starter default; Elasticsearch is the production scale-out option. We push search hard in Chapter 09.

— CONCEPT · NOBODY WIRES THOSE MODULES BY HAND

You start from hapi-fhir-jpaserver-starter.

It is a runnable Spring Boot application that wraps jpaserver-base, configured almost entirely through one application.yaml file — the hapi.fhir.* keys. We point at those keys here; Chapter 07 walks them.

— CONFIGURED BY APPLICATION.YAML

hapi.fhir.* — fhir_version · storage · partitioning · batch jobs · validation · CORS

You customize by adding your own application class, interceptors, and datasource beans on top. The keys exist now; turning the knobs is the next chapter.

2026 baseline

- HAPI 8.x
- Spring Boot 3
- Java 17+
- Jakarta namespace

— CONCEPT · FRAMEWORK ANCHOR — THE PATTERN WE USE

Do not edit the starter in place. Vendor it as an overlay.

— 1 · PULL UPSTREAM MERGEABLE

git subtree

We bring the upstream starter in with git subtree, so we can pull new HAPI releases as clean three-way merges.

— 2 · OUR CODE, OUR PACKAGE

com.dudoxx.fhir.r4 — package convention

All of our own code lives in an overlay source tree, under our own package convention — never mixed into vendored files.

— 3 · BRACKET UNAVOIDABLE EDITS

// DDX-OVERLAY-BEGIN … // DDX-OVERLAY-END

When we must touch a vendored upstream file, we bracket the change with two comment markers. A future subtree pull merges cleanly, and every local change is one grep away.

Ava → Marcus

That is the trap. The overlay also re-enables Clinical Reasoning, which the stock starter ships disabled.

— COMPARISON · WHY NOT JUST FORK IT?

Upstream-mergeable beats forked-and-frozen.

Many teams fork — and then they freeze. The overlay flips that trade. Here is the difference, side by side.

— HARD FORK — THE TRAP

You own the entire tree

Every upstream release becomes a manual reconciliation. The drift compounds, merges get painful, and eventually you stop upgrading at all.

— VENDORED OVERLAY — THE PATTERN

git subtree pull stays mergeable

Upstream three-way-merges against your subtree. Your edits stay isolated and auditable behind the markers, and you ride HAPI 8.x point releases instead of fearing them.

- **Fork — drift compounds, upgrades stall**
- **Overlay — clean merges, edits a grep away**

- **The takeaway for a server that moves as fast as HAPI**

— WALKTHROUGH · ONE REQUEST, END TO END

Every slide so far is one stop on this path.

A GET or a POST enters at the top and exits as serialized FHIR. The modules dispatch, the schema persists — this is the spine.

STEP 1 · DISPATCH

RestfulServer

The dispatcher receives the HTTP request.

STEP 2 · ROUTE

IResourceProvider

Routes to the provider for that resource type.

STEP 3 · PERSIST

DAO → Hibernate

The operation becomes SQL against the schema.

STEP 4 · LAND

HFJ_* tables

Rows written; FHIR serialized back out.

— ON THE WAY THROUGH

Interceptor Pointcuts fire — auth · partitioning · audit

Pointcuts are the hook surface where you inject cross-cutting behavior between dispatch and the DAO. We write our own in Chapter 10.

— DIAGRAM · THE WHOLE PICTURE, ON ONE SCREEN

None of it is magic once you see the layers.

A client sends FHIR over HTTP. Every box here is a slide we just did — assembled into a production FHIR server.

● CLIENT

FHIR over HTTP

REST request from any conformant client.

● SLIDES 05 · 06

Spring Boot starter + DDX overlay

jpaserver-starter with our overlay on top.

● SLIDES 02 · 08

HAPI module stack

RestfulServer dispatches; modules handle it.

● SLIDES 03 · 04

Hibernate → HFJ_*

Mapped down and persisted.

— PRIMARY STORE · SLIDE 03

Postgres — HFJ_RESOURCE + HFJ_RES_VER

Current state plus full version history, mapped by Hibernate ORM.

— FULL-TEXT SIDECAR · SLIDE 04

Lucene index

Hibernate Search alongside the relational store for _content and _text.

— TRY IT YOURSELF · UNDER A MINUTE

Clone the starter, boot it, and read its own software block.

In under a minute you have a live FHIR server on localhost. Then GET /fhir/metadata and pipe to jq — the running server names its own HAPI version in the CapabilityStatement software block.

— RUN THIS IN YOUR TERMINAL

```
git clone https://github.com/hapifhir/hapi-fhir-jpaserver-starter
cd hapi-fhir-jpaserver-starter && mvn spring-boot:run -Pboot
curl http://localhost:8080/fhir/metadata | jq '.software'
```

Recap

- 4 module layers
- HFJ_* schema + Hibernate
- Lucene full-text
- The overlay pattern