

Formal Architecture Specification v1

System: Healthcare EHR & Scheduling Platform

Architecture Style: Clean Architecture (Domain → Application → Infrastructure)

Audience: Engineering, Architecture, Compliance, QA

Status: Draft v1 (Authoritative Blueprint)

1. Architectural Objectives

This document defines the *non-negotiable foundation* for the system.

The architecture must guarantee:

- Clinical correctness (no invalid states possible)
- Financial consistency (no partial or corrupt transactions)
- Security by default (RBAC, ownership, auditability)
- Long-term maintainability
- Replaceable infrastructure (ORM, framework, queues, auth provider)
- HIPAA-aligned auditability and data isolation

Principle: Business rules live in the Domain. Frameworks are implementation details.

2. Architectural Layers

2.1 Dependency Rule

Dependencies always flow inward:

Infrastructure → Application → Domain

The Domain layer has **zero dependencies** on: - NestJS - Prisma - Express - Controllers - Databases - External services

3. Layer Responsibilities

3.1 Domain Layer (/domain)

Purpose: Pure business logic and invariants.

Contains: - Entities - Aggregate Roots - Value Objects - Domain Events - Repository Interfaces

Forbidden: - Prisma imports - NestJS decorators - HTTP concepts - DTOs - Controllers

3.2 Application Layer (`/application`)

Purpose: Use cases and orchestration.

Contains: - Use cases (e.g. CreateVisit, RecordPayment) - Application services - Ports (interfaces for infra) - Command/query handlers

Responsibilities: - Transaction orchestration - Authorization enforcement (policy level) - Calling repositories - Publishing domain events

3.3 Infrastructure Layer (`/infrastructure`)

Purpose: Implementation details.

Contains: - Prisma repositories - NestJS controllers - Guards, filters - Database adapters - External integrations

Responsibilities: - Mapping HTTP → Application DTOs - Mapping Prisma → Domain objects - Implementing repository interfaces

4. Aggregate Roots (Authoritative)

4.1 Patient Aggregate

Root: Patient

Purpose: Identity, demographics, consent, patient lifecycle

Owns: - Profile (name, DOB, contact, etc.) - Patient preferences - Consent flags

Invariants: - Patient must always be linked to exactly one User - Demographics cannot be mutated by anyone except patient/admin

Does NOT own: - Visits - Invoices - Appointments

References only by ID.

4.2 Appointment Aggregate

Root: Appointment

Purpose: Scheduling lifecycle

States: - REQUESTED - CONFIRMED - CANCELLED - COMPLETED

Owns: - Schedule time - Status transitions

Invariants: - Cannot transition from CANCELLED → CONFIRMED - COMPLETED appointments cannot be modified

Events: - AppointmentConfirmed - AppointmentCancelled - AppointmentCompleted

4.3 Visit Aggregate (Clinical Core)

Root: Visit

Purpose: Clinical truth container

Owns: - Procedures - MedicalRecords - Diagnoses - Notes

Invariants: - Visit must reference valid PatientId and DoctorId - Procedure cannot exist without Visit - MedicalRecord cannot exist without Visit (unless explicitly standalone)

Events: - VisitStarted - VisitCompleted - ProcedureRecorded - RecordAdded

4.4 Invoice Aggregate

Root: Invoice

Purpose: Financial correctness

Owns: - Payments - Balance - Status

States: - DRAFT - PARTIALLY_PAID - PAID - REFUNDED

Invariants: - Total paid cannot exceed total amount - PAID invoice cannot accept new payments - REFUNDED invoice must have prior PAID state

Events: - InvoiceIssued - PaymentRecorded - InvoiceSettled

5. Domain Events

All cross-aggregate communication must occur via events.

Examples:

Event	Trigger	Consumers
AppointmentCompleted	Appointment status → COMPLETED	Visit service creates Visit
VisitCompleted	Doctor closes visit	Invoice service generates invoice
PaymentRecorded	Payment added	Notification service, audit

Events are immutable facts.

6. Repository Contracts (Domain Interfaces)

Repositories live in:

domain/repositories/

Example:

- PatientRepository
- VisitRepository
- AppointmentRepository
- InvoiceRepository

Each repository exposes only domain language methods:

- findById()
- save()
- delete()
- findByPatient()

No Prisma, no SQL, no DB concepts.

7. Type System Strategy

Three distinct type systems:

7.1 Domain Types

- Pure business objects
- No decorators
- No validation libraries

7.2 DTOs (API Boundary)

- Exist only in Infrastructure

- Validated via class-validator
- Converted to Application commands

7.3 Persistence Models (Prisma)

- Exist only in Infrastructure
- Mapped into Domain objects via mappers

Never expose Prisma types outside Infrastructure.

8. Validation Strategy

Layer	Responsibility
DTO	Shape + format validation
Application	Authorization + intent validation
Domain	Invariant enforcement

Example: - DTO validates amount is number - Application validates user allowed to pay - Domain validates amount doesn't violate invoice rules

9. Authorization Placement

Authorization rules live in:

- Application layer (policy enforcement)
- Domain layer (invariants about ownership)

Guards only authenticate. They do not contain business rules.

10. Module Communication Rules

Forbidden: - Module A importing Module B service directly

Allowed: - Both depend on Application services - Both publish/consume domain events

This prevents tight coupling and circular dependencies.

11. Folder Structure (Target State)

```
src/
  domain/
    entities/
    events/
    repositories/
    value-objects/
  application/
    use-cases/
    services/
    ports/
  infrastructure/
    http/
      controllers/
      dtos/
    prisma/
      repositories/
      mappers/
  auth/
  logging/
modules/
  patient/
  appointment/
  visit/
  billing/
```

12. Aggregate Integrity Rules

Rule	Enforcement Layer
No procedure without visit	Domain
No payment exceeding invoice	Domain
No unauthorized access	Application
No invalid request shapes	DTO
No DB leaks to controllers	Architecture

13. Compliance Mapping (HIPAA-aligned)

Requirement	Covered By
Least privilege	RBAC + ownership rules
Auditability	AuditLog + domain events
Data isolation	Aggregate boundaries
Tamper resistance	Domain invariants
Traceability	Event-driven history

14. Architectural Guarantees

If this document is followed, the system guarantees:

- No accidental business rule violation
 - No hidden coupling between modules
 - No framework lock-in
 - Clear onboarding for new engineers
 - Safe extension of new workflows
 - Auditable reasoning for compliance
-

15. Status

This document is now the **canonical foundation**.

No structural implementation should occur without aligning to this spec.

Next step after approval:

Migration Plan — mapping current codebase → target architecture safely, incrementally, without breaking functionality.