

Chapter 8 / Capítulo 8

Primum non nocere: Software engineering
for medicine
ISBN: 978-9915-9876-8-2
DOI: 10.62486/978-9915-9876-8-2.Ch08
Pages: 133-150



© 2026 The authors. This is an open access
book chapter distributed under the terms of
the Creative Commons Attribution (CC BY)
4.0 License.

Chapter 8. Architecture in the Healthcare Environment

Annier Jesús Fajardo Quesada  

Eduardo Antonio Hernández González 

René Herrero Pacheco 

Rolando Javier Álvarez Pérez 

1 Granma University of Medical Sciences. Granma, Cuba.

2 University of Medical Sciences of Pinar del Río. Pinar del Río, Cuba.

“Software architecture is not an abstract design exercise but rather the structure that determines how a system will respond when it is needed most. In a hospital, where every second counts and every failure has consequences, choosing the right architecture is not just another technical decision: it is a clinical decision with a capital C.”

Imagine a hospital information system that manages appointments, medical records, prescriptions, lab results, images, billing, and dozens of other features. For years, this system has operated as a monolith: a single application that does it all, with a central database, and a development team that maintains the whole system. One day, someone proposes changing how appointments are managed, adding artificial intelligence features to optimize the schedule. However, making that change requires modifying the entire monolith, with the risk of affecting other parts of the system. The team hesitates, deadlines are pushed back, and the improvement never materializes. This scenario, which is common in many hospitals, illustrates the fundamental dilemma of software architecture in healthcare settings: How do we balance the need for stability and reliability with the need to evolve and innovate?

Software architecture is the discipline concerned with the structure of systems—how their components are organized, how they relate to one another, and how they evolve over time. In the healthcare sector, architectural decisions have implications that go beyond the technical: they affect the ability to integrate with other systems, the ease of incorporating new features, resilience against failures, and ultimately, patient safety. That is why it is so important for both developers and clinicians to understand the fundamentals of architecture—not to make technical decisions but to participate in the conversations that will determine what the systems they use every day will look like.

In this chapter, we explore the key architectural decisions faced by teams building healthcare software. We begin with the classic debate between monoliths and microservices, analyzing what makes sense in a hospital setting. We will continue with the choice between cloud computing and edge computing, considering the implications for latency, privacy, and processing. We address the particularities of clinical databases, which must handle sensitive information with very demanding availability and traceability requirements. Finally, we will reflect on how to design systems with interoperability in mind from the start rather than as a later addition. This is not an exhaustive technical guide but rather an overview that allows readers to understand the options and their implications.

8.1 MONOLITH VS. MICROSERVICES: WHAT MAKES SENSE HERE

There is a recurring and sometimes heated debate in the world of software development that pits two radically different architectural philosophies against each other, each with its own traditions, tools, and battlegrounds. On the one hand, monoliths: single applications that concentrate all functionality into a single executable block, with a single central database, and a unified, sequential development, testing, and deployment process. On the other hand, microservices are sets of small, independent, and autonomous applications, each with its own dedicated database, that communicate with one another via network APIs and that can be developed, deployed, and scaled separately, even by different teams. Each approach has its own staunch advocates who proclaim its absolute virtues and its detractors who point out its fatal flaws, and the choice between one or the other can profoundly shape a system's entire future development, its operating costs, and its capacity for evolution. In the healthcare sector, however, this choice takes on additional nuances of criticality, regulation, and operational context that should be analyzed carefully and without dogmatism imported from other, less regulated sectors.

Monoliths have a long and venerable tradition in hospital software, and this is neither by chance nor due to technological conservatism. The large electronic health record systems that dominate the global market, such as Epic, Cerner, and Meditech, are essentially monolithic architectures: massive, mature applications that integrate hundreds of features into a single, coherent technological environment. This approach offers clear operational advantages that explain its enduring popularity. The first is systemic consistency: since all code and data reside within the same logical system, the end-user experience is uniform and predictable, data are centralized without synchronization issues, and functionalities are integrated naturally through internal program calls rather than through complex network orchestrations. The physician navigates a familiar environment where the prescription, laboratory results, diagnostic imaging, and discharge report share the same interface, the same database, and the same security model.

The second advantage is relative operational simplicity: managing a single monolithic system, even a large one, is conceptually simpler than managing dozens of independent services that interact in complex ways. The backups are from a single database, disaster recovery plans focus on a single point, and performance monitoring has a single target. The third advantage, crucial in clinical settings, is latency performance: internal communications between modules within a monolith are carried out via calls to functions in shared memory, which are orders of magnitude faster than HTTP calls to remote services—a factor that can be decisive in situations where the speed of information access is critical for making urgent medical decisions.

However, monoliths also carry significant disadvantages that have become more evident and costly over time and as expectations have evolved. The first is structural rigidity: modifying any part of the system, even for localized improvement, requires touching the whole, recompiling, thoroughly testing the entire system, and deploying in bulk, with the inherent risk of seemingly unrelated functionalities being affected. This makes innovation slow, costly, and feared because the cost of an error in production is systemic. The second drawback is granular scalability: if a specific feature, such as querying lab results during an epidemic outbreak, requires more computational resources, the entire monolith must be vertically scaled, even if the rest of the modules do not need it, which is economically inefficient. The third is limited resilience: a catastrophic failure anywhere in the code, such as a memory error or an unhandled exception, can reduce the entire monolith, affecting all functionalities simultaneously and leaving the hospital without an information system. In a healthcare environment where 24/7 availability is critical for patient safety, this is particularly problematic and feared.

8.1.1 THE PROMOTION OF MICROSERVICES AND THEIR RISKS

Microservices emerged precisely in the broader technology industry to address these structural limitations of large, mature monoliths. Instead of a single massive application, many small, specialized applications are built, each responsible for a specific business function: patient demographic management, appointment scheduling, electronic prescribing, results viewing, billing, etc. Each microservice has its own database optimized for its use case, its own potentially autonomous development team, and its own independent deployment lifecycle. They communicate with each other via well-defined APIs, typically using HTTP REST and formats such as JSON or Protobuf, over an internal network or even the internet. This architecture offers significant advantages in terms of flexibility for evolution, independent scalability, and resilience to partial failures.

Flexibility is perhaps the advantage most valued by agile development teams. With microservices, a component can be modified, refactored, or even completely replaced without affecting the rest of the ecosystem, provided that the API contract is respected. A hospital's team might decide to modernize its appointment management system by hiring a specialized team to develop a new microservice from scratch using current technologies, while the rest of the system continues to operate stably with legacy components. This allows for continuous, gradual evolution without the need for costly and risky full-system migrations. It also facilitates the adoption of new technologies without the accumulation of technical debt: a microservice can be implemented in a modern language such as Go or Kotlin or a framework such as Node.js—different from the rest of the ecosystem—if that provides productivity or performance benefits for that specific use case.

Independent scalability is another key operational advantage. If a specific feature—such as the bulk querying of lab results during a pandemic where thousands of tests are performed daily—experiences extraordinary spikes in demand, additional instances of that specific microservice can be automatically deployed using orchestrators such as Kubernetes, without the need to vertically scale the entire monolithic system. This allows for a much more efficient and cost-effective use of infrastructure resources, especially in public cloud environments where one pays strictly for consumption, and facilitates adaptation to unpredictable future growth, since each component can scale horizontally according to its own demand curves.

Resilience is perhaps the most significant advantage from a purely clinical and patient safety perspective. In a well-designed microservice architecture following patterns such as bulkheads and circuit breakers, the catastrophic failure of one component should not propagate or affect the other services. If the scheduled appointment management microservice decreases because of a coding error, patients can still be seen in the emergency department, and the emergency admission service continues to function perfectly because they are independent services. If the electronic prescription service experiences a temporary outage, laboratory results

remain available for review, and imaging reports can still be viewed. This ability to isolate failures and limit the “surface area” of an incident is crucial in an environment where the continuous availability of certain functionalities can literally be a matter of life and death. Naturally, there are functionalities that maintain critical dependencies on one another, and a failure in a truly central service can have serious systemic consequences, but a good microservice architecture intentionally minimizes those hard dependencies through decoupling patterns.

However, microservices are not a universal magic solution, and their own significant drawbacks are often underestimated in the initial adoption phase. The main problem is explosive operational complexity. Managing dozens or even hundreds of independent services in production, each with its own database, its own environment configurations, its own potentially asynchronous deployment cycles, and its network interdependencies, is orders of magnitude more complex than managing a monolith, even a large one. Sophisticated and mature tools are required for container orchestration, dynamic service discovery, centralized distributed monitoring, aggregated log management, and handling of cascading failures. A platform team with advanced skills in these areas is needed, along with a reliability engineering culture that many traditional healthcare organizations simply do not possess. For small organizations or those with limited IT resources, this complexity can be prohibitive and create more problems than solutions.

Another inherent drawback is the introduced network latency. Communications between microservices, even on a fast internal network, occur via network protocols that introduce millisecond-level delays that do not exist in internal process calls within a monolith. In most administrative use cases, these delays are perfectly tolerable, but in real-time critical applications—such as continuous patient monitoring in intensive care units where responses are required in fractions of a second—they can be problematic or even unacceptable. Data consistency must also be considered: Since each microservice has its own isolated database, maintaining transactional consistency between them requires the implementation of additional complex patterns, such as distributed sagas or eventual consistency with trade-offs, which add considerable complexity to the design and are difficult to debug when they fail.

8.1.2 CONTEXTUAL DECISIONS AND HYBRID ARCHITECTURES

In the specific context of the healthcare environment, which architectural approach makes the most sense? The honest answer, as is often the case in mature software engineering, is that it depends critically on multiple contextual factors. It depends on the size and complexity of the organization, the available technical and human resources, the criticality and diversity of the applications to be supported, the required pace of innovation, applicable regulatory constraints, and many other organizational and cultural factors. What is absolutely clear is that there is no single, universally optimal solution and that the decision must be made after a careful, multidisciplinary analysis of the specific circumstances, not by following technological fads or vendor pressures.

For relatively small information systems, such as those of a medium-sized regional hospital, a specialized clinic, or a primary care center, a well-designed monolith remains perfectly adequate and is often preferable. Operational simplicity, user consistency, and the reduced need for massive scalability make the disadvantages of monoliths less relevant, while the complexity of managing microservices would be an unnecessary burden. Furthermore, limited IT resources in these environments make the cognitive overhead of microservices difficult to bear without risk. In these cases, the sensible choice is to opt for a monolith, but one designed internally in a modular fashion, with well-defined interfaces between its logical components, following clean architecture principles, to facilitate a potential future migration to microservices should the organization's needs change radically.

For very large and complex systems, such as those of large university hospitals with multiple campuses, integrated regional healthcare networks, or national digital health platforms, microservices can offer significant strategic advantages that justify the investment in complexity. The ability to evolve components independently, to scale selectively according to actual load requirements and to isolate failures to maintain continuity of care amply justifies the investment in platform capabilities and specialized talent. However, it is crucial to address this complexity systematically and progressively, investing seriously in the necessary orchestration, observability, and security tools, and designing the architecture carefully from the outset using proven patterns. Chaotic or rushed migration to microservices, if poorly executed, can be worse operationally than a well-managed, mature monolith.

Pragmatic hybrid approaches may also be suitable for many intermediate scenarios. For

example, maintaining a stable monolithic core containing the most critical, mature, and stable functionalities of the clinical business and implementing peripheral functionalities—those that change frequently, experience variable load spikes, or require different specific technologies—as independent microservices. Alternatively, using a “minimodules” or macroservice architecture within a logical monolith, which allows for some fault isolation and independent deployments, facilitates a potential full migration to microservices if conditions warrant it in the future. The important thing is not to dogmatize or idealize any architecture but to evaluate each specific case using technical and clinical criteria and to make the decision that best serves the present and foreseeable future needs of the organization and its patients.

For clinicians and healthcare professionals involved in these architectural debates, understanding the practical implications of technology choice helps them better advocate for their operational and patient needs. If rapid innovation and experimentation with new features are strategically important for the hospital, they can support architectural choices that facilitate agile evolution with sound arguments. If, on the other hand, operational stability, predictable performance, and maximum availability are critical for patient safety, they can tip the balance toward more conservative and proven options. Your expert voice in the clinical domain is absolutely essential to ensure that technical decisions are not solely based on abstract software development criteria or industry trends but also on real clinical needs, existing workflows, and potential risks to patient care.

One aspect that is often overlooked or underestimated in the simplistic monolith versus microservice debate is data governance and privacy. In a traditional monolith, the central database makes it easier to maintain absolute transactional consistency and enforce uniform access control, auditing, and privacy policies at a single point. In a microservice architecture with distributed and potentially heterogeneous databases, governance becomes inherently more complex and requires explicit coordination. It is essential to ensure that each microservice consistently implements the same organizational policies for protecting sensitive data, that identifiable data are protected across all repositories and that a consistent, global access log is maintained for regulatory auditing. This requires the implementation of an additional layer of security infrastructure and distributed policy tools, such as service mesh sidecars, which add complexity but are necessary for compliance.

It shows that many organizations that initially embraced microservices with enthusiasm, drawn by the promises of agility, end up facing unexpected problems of operational complexity, rising infrastructure costs, and difficulties in finding qualified talent. Some consciously choose to return to more monolithic approaches or, more frequently, to intermediate solutions that have come to be called “modulitos”: a pragmatic solution that seeks to capture the best of both worlds, maintaining operational simplicity where possible and flexibility where necessary. The general lesson is that there are no universal architectural magic formulas and that technical humility, adaptation to the specific circumstances of each organization, and iterative evolution are more important than blindly following the architectural fads of the moment.

Software architecture is a means to construct information systems that effectively help clinicians care for their patients better and managers operate efficiently. A well-designed, documented, tested, and operated monolith that runs reliably 24 hours a day and allows for the necessary business evolution is infinitely better than a complex and fragile microservice architecture that no one in the organization fully understands, that generates constant incidents and that consumes the entire IT budget on platform maintenance. Conversely, a well-implemented microservice architecture with trained teams, appropriate tools, and mature processes—one that enables rapid innovation in response to emerging clinical needs and elegantly isolates failures to maintain continuity of care—is better than a rigid, obsolete monolith that prevents any improvement and must be restarted weekly. What truly matters is maintaining an obsessive focus on tangible clinical value and choosing or evolving the architecture that best serves that purpose in each specific context of time, place, and organization.

8.2 CLOUD VS. EDGE: LATENCY, PRIVACY, AND PROCESSING

Imagine an intensive care unit where dozens of critically ill patients are simultaneously connected to sophisticated multiparameter monitors that continuously record, several times per second, their heart rate, invasive blood pressure, pulmonary oxygen saturation, core temperature, and dozens of other essential vital signs. At every moment, these monitors generate a torrent of data that must be immediately analyzed by algorithms to detect arrhythmias, trends in

hemodynamic deterioration, and patterns of incipient sepsis to trigger smart alarms when necessary and alert nursing staff before the patient goes into cardiac arrest. If that critical data had to travel first through the hospital network, then to the internet, be processed on a remote server in the public cloud, wait its turn in a shared processing queue, and finally send an alert back to the bedside monitor, the cumulative delay could be several seconds or even tens of seconds under network congestion conditions. In a sudden cardiac arrest situation, where every second without intervention drastically reduces the chances of intact brain survival, that delay is absolutely unacceptable and potentially lethal.

On the other hand, let us now imagine an ambitious multicenter epidemiological study that needs to analyze health trends in millions of patients over several decades, identify risk factors for rare diseases, evaluate the population-level effectiveness of large-scale health interventions, or train predictive artificial intelligence models with massive datasets. In that case, cloud computing, with its virtually unlimited capacity for distributed storage, massively parallel computing, and access to specialized resources such as high-density GPUs, is the only technically viable and economically reasonable option. These two extreme but realistic scenarios illustrate the fundamental and seemingly contradictory dilemma between cloud computing and edge computing in the modern healthcare environment and the need for an architecture that does not choose between one or the other but rather combines both paradigms according to the specific clinical context.

Cloud computing consists fundamentally of processing and storing data in remote, massively scalable data centers that are accessible via the internet through standardized APIs. (1) The advantages of this model are evident and have driven its widespread adoption across all sectors: virtually unlimited horizontal scalability that automatically adapts to demand, massive processing capacity for intensive workloads, ubiquitous access from anywhere with connectivity, pay-as-you-go economic models that avoid initial capital investments in proprietary infrastructure, and the ability to delegate infrastructure management to specialized providers. Major cloud providers such as Amazon Web Services, Microsoft Azure, and the Google Cloud Platform offer specific services designed for the healthcare sector, with regulatory compliance certifications such as HIPAA BAA or GDPR, and specialized tools for the secure processing of structured and unstructured clinical data. For applications in epidemiological research, population-based public health, healthcare administrative management, or retrospective analysis of large cohorts, the cloud is the natural and almost unavoidable choice.

Edge computing, on the other hand, represents the opposite yet complementary architectural philosophy: it involves processing data physically close to where it is generated, at the “edge” of the network, before it travels to the central cloud. (2) This processing can occur on the medical device itself, at a local aggregation gateway, or on a hospital server located within the organization’s perimeter. The main and most frequently cited advantage is extremely low latency: since the data do not have to be sent through multiple network hops to the internet and back, processing and response are practically instantaneous, on the order of milliseconds. There are also critical advantages in terms of privacy and data sovereignty: the most sensitive health information can be processed and stored locally without ever leaving the hospital’s controlled and regulated environment, reducing the surface area exposed to external security breaches. Finally, there are advantages in terms of operational autonomy: the system can continue to function critically even if the internet connection fails completely, maintaining continuity of care in emergency situations. For real-time physiological monitoring applications, control of active medical devices such as infusion pumps, or situations where external connectivity is not guaranteed for geographical or security reasons, edge computing is technically irreplaceable.

8.2.1 CRITICAL FACTORS IN ARCHITECTURAL DECISION-MAKING

Network and processing latency is perhaps the most critical—and often decisive—factor in choosing between the cloud and the edge for a specific use case. In strict real-time applications, such as continuous patient monitoring in intensive care units, closed-loop control of smart infusion pumps that automatically adjust doses on the basis of the patient’s response, or computer-assisted guidance for complex surgical procedures, delays of even a few hundred milliseconds can be unacceptable and can compromise patient safety. Transmission to the cloud inevitably introduces variable and hard-to-guarantee latencies, depending on the momentary quality of the connection, the geographical distance to the data center, the load from other users on the shared service, and internet congestion. In these mission-critical cases, edge processing is the only viable

option from a medical systems engineering standpoint. However, not all healthcare applications require strict real-time performance; for many routine clinical operations, such as reviewing stable patients' medical records, generating administrative reports, or viewing nonurgent laboratory results, latencies of one or several seconds are perfectly tolerable and do not impact the quality of care.

Data privacy and sovereignty are key factors that heavily influence architectural decisions, especially in jurisdictions with strict regulations such as the General Data Protection Regulation (GDPR) and the European General Data Protection Regulation (GDPR). Healthcare data are considered a special category of highly sensitive personal data, and sending them to the public cloud inherently involves trusting that the cloud provider will adequately protect them, comply with applicable regulations, and not use them for unauthorized secondary purposes. Major cloud providers offer robust contractual guarantees, with certifications such as the HIPAA in the United States and the ISO 27001 and GDPR compliance mechanisms in Europe, including standard contractual clauses and certifications of responsible conduct. However, for some organizations—especially those in environments with very strict sector-specific regulations, those handling particularly vulnerable patients such as VIPs or sensitive personnel, or simply due to a conservative organizational culture—the option of keeping sensitive data within the hospital's physical and logical perimeter may be preferable or even mandatory. Edge computing allows sensitive data to be processed locally, sending only information that has been previously anonymized, statistically aggregated, or stripped of identifiers to the public cloud when necessary for backup or population analysis functions.

Raw processing capacity is another technical factor to be carefully considered in the design process. The cloud offers virtually unlimited and elastic computing power and is capable of running complex artificial intelligence algorithms that require high-density GPUs, massive distributed processing clusters, or object storage at the petabyte scale. In contrast, the edge has resources inherently limited by the size, weight, power consumption, and cost constraints of local devices: a bedside monitor or a home gateway cannot run deep learning models with millions of parameters or store massive databases. However, this technological gap is rapidly narrowing: new processors specifically designed for artificial intelligence at the edge, such as neural processing units (NPUs), allow increasingly complex models to be run directly on medical devices, and advanced techniques such as model compression, quantization, or federated learning enable processing to be intelligently distributed between the edge and the cloud according to the capabilities of each level. The technological trend is toward a fluid continuum, where decisions about where to process each specific task are made dynamically and adaptively on the basis of the latency, privacy, and capacity requirements at any given moment.

Network connectivity is often underestimated but is a critical factor in planning resilient healthcare architectures. In hospital settings, even in advanced urban centers, high-speed, low-latency internet connectivity is not guaranteed at all times. There may be planned outages due to infrastructure maintenance, technical issues with the service provider, or even sophisticated cyberattacks that leave the hospital without external access during critical hours. Medical applications that are truly critical to patient safety must be able to continue operating at full capacity under these conditions of network isolation, which architecturally requires substantial local processing and storage capabilities at the edge. A patient monitoring system that relies exclusively on the cloud to generate vital sign alarms can fail catastrophically precisely when it is needed most—during a cyberattack that paralyzes external connectivity. That is why it is important to design medical systems with an intelligent degradation mode: systems that operate with reduced but essential functionality when the connection fails, that store data locally in a queue and that automatically synchronize when connectivity is restored without loss of critical information.

The massive volume of data generated by modern medical devices is another factor tipping the balance toward hybrid architectures. Contemporary medical devices generate enormous amounts of data: high-resolution images from advanced modalities, video sequences of endoscopic or surgical procedures, and continuous high-frequency recordings of physiological signals such as ECG or EEG. Sending all these raw data to the cloud can be prohibitively expensive in terms of network bandwidth, generate unacceptable latencies, and require massive cloud storage with high recurring costs. The edge enables preprocessing, intelligent filtering, lossless compression, or local analysis to extract only the relevant and condensed information, sending only the processed results, detected anomalies, or statistically aggregated data to the cloud. For example, a smart surveillance camera monitoring a patient at risk of falling can locally analyze video images using computer vision and send to the cloud only the short clips where it detects

anomalous movement or a fall, rather than continuously streaming hours of video without events. This edge filtering approach drastically reduces network bandwidth and cloud storage requirements while maintaining essential clinical functionality.

8.2.2 FOG ARCHITECTURE AND CLOUD-EDGE SYMBIOSIS

The intelligent combination of cloud and edge, technically known as fog computing, is rapidly gaining ground in sophisticated healthcare environments as the dominant architectural choice. In this hierarchical model, first-tier edge devices perform immediate, critical, low-latency processing; second-tier local gateways aggregate data from multiple neighboring devices, perform intermediate trend analysis, and coordinate communication with higher tiers; and the central cloud handles long-term persistent storage, massive population analysis, AI model training, and global coordination across multiple hospitals or regions. This well-defined layered architecture allows each technological level to be optimized for its specific function, elegantly balancing the conflicting requirements of minimal latency, maximum privacy, and unlimited processing capacity.

A concrete, realistic example can help us better understand this practical combination. Imagine a comprehensive remote home monitoring program for patients with established chronic heart failure. At the patient's home, a purpose-built edge gateway continuously receives data from a connected smart scale, an automatic wrist blood pressure monitor, and an activity and heart rate monitor. This gateway locally analyzes trends in weight, blood pressure, and heart rate variability using programmed clinical rules, and if it detects patterns indicative of incipient cardiac decompensation, it immediately sends a high-priority alert to the patient's phone and simultaneously to their primary care center—all without leaving the local environment and in a matter of fractions of a second. Daily, during times of lower network load, the gateway sends a structured summary of all the day's data to the regional healthcare cloud for automatic incorporation into the patient's electronic health record. Monthly, the cloud runs a sophisticated artificial intelligence algorithm that analyzes the longitudinal progression of all patients in the program, identifying subtle predictive patterns that enable proactive improvements to alert protocols for the entire cohort. Each level of the architecture does what it does best, in the most appropriate place, with the most efficient resources.

For developers of modern healthcare software, the decision between the cloud and edge is no longer an exclusive binary choice but rather involves a rich continuum of architectural options that must be evaluated on a case-by-case basis. Each specific functionality, data flow, and clinical use case must be analyzed to determine its requirements in terms of maximum acceptable latency, required privacy level, connectivity availability, and necessary processing capacity. On the basis of this multidimensional analysis, the most appropriate architecture is designed, which in practice almost always combines both approaches in varying proportions. Modern development tools greatly facilitate this hybridization, with platforms such as Kubernetes that allow the same code to be deployed across different edge or cloud environments and data synchronization frameworks that manage consistency across layers transparently for the application developer.

For clinicians and healthcare professionals involved in these technology projects, understanding the practical implications of choosing between cloud and edge helps them better assess technical decisions that often seem opaque or arbitrary to them. When a developer explains that a certain critical alert functionality cannot be implemented purely in the cloud because of insurmountable network latency issues or that certain particularly sensitive genomic data cannot physically leave the hospital for ethical and regulatory privacy reasons, the clinician can understand the real technical limitations and collaborate constructively in seeking viable architectural alternatives. It can also help prioritize intelligently among multiple competing needs: which clinical features are truly critical in strict real time versus which can tolerate a few seconds of delay? Which data are so sensitive that they must always remain under absolute local control? Their expert perspective on the clinical domain is absolutely essential for making balanced architectural decisions that do not sacrifice patient safety on the altar of technical efficiency.

Cybersecurity is a cross-cutting issue that critically affects both cloud and edge deployments, albeit with different risk profiles. In the cloud, security responsibility is shared according to the shared responsibility model: the cloud provider secures the underlying infrastructure, the hypervisor, and physical storage; the healthcare organization secures its own applications, configurations, data, and access policies. At the edge, the organization has full control and absolute sovereignty over the infrastructure, but it also assumes full responsibility for implementing,

maintaining, and updating all appropriate security measures. In both cases, strong encryption of data in transit between layers and at rest in storage is needed, along with identity- and role-based access control, comprehensive auditing of access logs, and proactive protection against external threats through intrusion detection systems. The choice between the cloud and edge should not be based on simplistic assumptions about which is inherently more secure but rather on a detailed and up-to-date analysis of the specific security capabilities of each option and the protection needs of each type of data.

The total economic cost of ownership is another factor that must be considered comprehensively and over the long term. The cloud offers an operational pay-as-you-go model that can be highly efficient for variable and unpredictable workloads but can prove surprisingly expensive for continuous massive volumes of data or nonstop intensive processing. The edge requires an initial capital investment in proprietary hardware, physical maintenance, and cyclical renewal, but variable operational costs are lower once the investment is amortized. A truly informed cost analysis must consider not only visible direct expenses but also the opportunity cost of being unable to innovate quickly because of architectural rigidity or the potentially catastrophic cost of a security failure or data breach that exposes the organization to millions of regulatory fines and irreparable reputational damage. It is a complex, dynamic calculation specific to each organization that must be reviewed periodically.

The technological future points unequivocally toward an increasingly sophisticated and transparent integration of the cloud and edge, with distributed artificial intelligence that continuously learns from the aggregated data of millions of patients in the central cloud but executes inferences and predictions locally in real time on the patient's edge devices. Machine learning models are trained centrally using large volumes of historical data in the cloud, achieving high diagnostic accuracy, but are then deployed locally optimized for immediate use at the point of care. Model updates are distributed continuously and securely, enhancing device intelligence without the need to replace physical hardware. This evolutionary symbiosis between the cloud and edge is likely to be the dominant architecture in digital health in the near future and is already emerging in the most advanced systems.

8.3 CLINICAL DATABASES: CHARACTERISTICS AND SPECIFIC FEATURES

Let us imagine for a moment the operational database of a medium-sized university hospital, a complex organization that handles hundreds of thousands of clinical episodes annually. It simultaneously houses the demographic data of hundreds of thousands of unique patients accumulated over decades; the detailed records of millions of outpatient consultations and emergency room visits; the results of tens of millions of laboratory tests of all kinds—from simple blood tests to complex genetic studies; structured reports and images from hundreds of thousands of diagnostic imaging studies; prescriptions for countless drug treatments with their dosages, routes of administration, and durations; and narrative clinical notes from generations of physicians documenting their reasoning, observations, and decisions. This database is not merely large in terms of storage volume; it is profoundly complex in its semantic structure, radically heterogeneous in its data types, and above all, it is absolutely critical to the continuity of healthcare operations. Clinical decisions that directly affect people's health—and sometimes their lives—depend on their continuous, proper functioning. If the database decreases because of a technical failure, hospital operations are ground to a halt within minutes, and patients in the emergency room cannot be properly cared for. If data are lost because of corruption or human error, the accumulated clinical information is irreparably destroyed, and the continuity of care is disrupted. If it is slow or unresponsive, doctors become frustrated, patients wait unnecessarily, and the quality of care visibly deteriorates. Designing, implementing, and managing a clinical database of this nature is not a trivial technical task; it requires a deep understanding of its specific characteristics and respecting them in every architectural decision.

The first distinctive and fundamental characteristic of clinical databases is the deeply semantic nature of the data they store, which transcends mere numerical or textual representation. They are not simply isolated numbers and texts but medical information with a wealth of context and exceptional meaning. A laboratory result, for example, is not just an abstract numerical value; it is a value with specific units of measurement that must be interpreted correctly, with normal ranges that depend critically on the patient's age, sex, and sometimes ethnicity, with clinical interpretations that vary according to the physiological and pathological context in which it was obtained. A narrative clinical note is not just a block of free-form text; it is a linguistic construct

containing complex implicit information, the physician's diagnostic reasoning, specific treatment plans, and sometimes uncertainties expressed in natural language that require intelligent processing to be exploited. A medical image is not just a large binary file; it is a structured study comprising multiple series acquired at different times or under different conditions, with technical metadata essential for its correct interpretation and with structured annotations that pinpoint specific findings. Clinical databases must be capable of representing, preserving, and enabling the retrieval of this multidimensional semantic richness in a way that allows it to be efficiently processed by both automated systems and healthcare professionals.

An absolutely nonnegotiable characteristic is the extreme sensitivity of the stored data from an ethical, legal, and social perspective. Clinical information is considered the most personal and sensitive category of personal data in virtually every jurisdiction worldwide, and its unauthorized disclosure—whether accidental or malicious—can have devastating and lasting consequences for affected patients, such as employment discrimination due to preexisting health conditions, social stigmatization due to mental or infectious diseases, family problems due to the disclosure of hereditary genetic information, or even risks to physical safety in the case of public figures.

For this reason, clinical databases must inherently and mandatorily incorporate robust, multidimensional security measures: strong encryption of data at rest to protect against physical theft of storage media, granular access control based on roles and clinical contexts to ensure that only legitimate authorized personnel access specific information necessary for their role, detailed and immutable auditing of all read and write operations to enable subsequent investigations, and technical mechanisms for anonymization and pseudonymization when data must be used for secondary purposes such as research or healthcare planning. This is not merely a technical implementation issue but also a legal, ethical, and social trust imperative. Noncompliance with applicable data protection regulations, such as the European GDPR or the U.S. HIPAA, can result in millions of dollars in fines for the institution, but worse still from a sustainability perspective, the loss of patient trust can be irreparable and destroy the social legitimacy of the healthcare system.

The urgent need for continuous, uninterrupted availability—the famous “five nines” or 99.999% uptime, which is actually insufficient for clinical criticality—is another key characteristic. A modern hospital never closes, not even for scheduled database maintenance. Emergencies continue to arrive at any hour of the day or night, operating rooms continue to perform scheduled and emergency surgeries, and intensive care units continue to monitor critically ill patients 24 hours a day, 365 days a year, without pause. The clinical database must be available and operational at all times, with fault-tolerant architectures that ensure that even in the face of serious technical incidents, service is maintained or restored in seconds, not hours. This inevitably requires high-availability architectures with active redundancy, continuous real-time backups, thoroughly tested disaster recovery plans, and proactive monitoring systems that are alert to any anomaly before it becomes a catastrophic failure. The cost of a prolonged outage is measured not only in lost administrative work hours or uninvoiced revenue but also potentially in human lives lost because of delayed care or a lack of critical information.

In a clinical database, information must be accurate at its source, complete in its recording, and reliable in its retrieval. An error in clinical data—such as a miscoded allergy, a lab result attributed to the wrong patient due to an identification error, or an incorrectly recorded medication dose—can directly lead to a serious diagnostic error, potentially lethal mistreatment, or avoidable harm to the patient. For this reason, clinical databases incorporate strict input validation mechanisms, referential integrity constraints that maintain consistency between related tables, and immutable audit logs that allow tracking of any changes made—when, by whom, and why. They must also elegantly handle complex situations such as correcting errors detected after the fact: if erroneous data are identified and corrected, it is not enough to simply overwrite it and lose the trail; the record of the original erroneous value, the correction applied, and the documented reason for the correction must be maintained because this history may be relevant in a legal context of a claim or a retrospective clinical investigation.

In medicine, knowing precisely who did what, exactly when, and for what documented reason is essential for safety, quality, and accountability—this is traceability. A drug prescription must be irrevocably linked to the identified prescribing physician, the patient receiving it, the exact time of the order, and the clinical indication that justified it. A change to a primary diagnosis must leave an immutable record of who made the change, with what reasoned clinical justification, and what the previous value was. Access to sensitive data regarding a famous patient or politician must be permanently logged for potential audits of legitimacy. Clinical databases must support this comprehensive level of traceability, which implies not only storing the current valid data but also maintaining the complete history of temporal changes, the operational metadata for each

transaction, and the explicit relationships between data and identified individuals.

A typical modern clinical database simultaneously stores strictly structured data—such as numerical laboratory values and their metadata—semistructured data—such as medical reports with predefined fields but variable narrative content—and completely unstructured data—such as free-form clinical notes in natural language, narrative discharge summaries, or voice recordings. It also stores massive binary objects such as DICOM radiological images, continuous physiological monitoring signals, video files of endoscopic procedures, or PDF documents of signed informed consent forms. Managing this heterogeneity of formats, sizes, and access patterns efficiently requires different types of physical and logical storage, as well as different indexing, compression, and query strategies optimized for each case. Traditional relational databases are necessarily complemented by NoSQL-type document stores, distributed file systems for large objects, and specialized databases for time series of physiological signals, all of which are integrated under an abstraction layer that presents a unified view to the user and applications.

Queries to clinical databases can vary extraordinarily in their patterns. A doctor in the emergency room may need to retrieve the complete medical history of an unknown patient who arrives unconscious in less than a second to make immediate, life- y decisions. An epidemiological researcher may run a complex analytical query that scans and aggregates millions of historical records over several minutes for a cohort study. An automated clinical alerting system may need to continuously monitor new data arriving from hundreds of patients simultaneously in real time to detect patterns of deterioration. The database must be capable of handling this variety of concurrent workloads without degrading the performance perceived by any user, which requires a careful, multifaceted design of optimized indices, horizontal and vertical partitioning strategies, and intelligent cache layers that anticipate frequent access patterns.

Clinical databases do not exist or operate in a technological vacuum; they must continuously exchange information with other systems within the same hospital, with external healthcare facilities for continuity of care, directly with patients via health portals, and with healthcare providers in the ecosystem. This involves natively supporting exchange standards such as HL7 v2 for messaging, FHIR for modern APIs, or DICOM for imaging; exposing well-documented programming interfaces for data querying and updating; and maintaining eventual consistency when the same information is replicated or referenced across multiple heterogeneous systems. Interoperability is not an optional technical add-on that can be postponed but a fundamental functional requirement for the operational reality of modern digital health.

Healthcare systems undergo organizational change, medical knowledge advances and alters what is considered relevant to records, and information needs evolve with clinical practice and regulation. A clinical database must be capable of adapting to these inevitable changes without losing or invalidating previously stored historical information, which holds legal, clinical, and research value. When a new data field is introduced because of a change in clinical practice, older records must remain accessible and interpretable. When a standard coding scheme is modified because of an update in a medical classification, historical data coded with previous versions must be correctly interpreted within their temporal context. This requires the implementation of strategies for managing flexible database schemas, data versioning mechanisms, medical vocabularies, and sometimes complex migrations that preserve historical semantics.

The volumes of healthcare data generated are growing steadily and rapidly: more diagnostic tests per capita due to defensive medicine and increased use of technology, images with higher spatial and temporal resolution due to technological advances, more connected devices generating continuous monitoring streams, and the emergence of massive amounts of genomic data that multiply the size of the record per patient by orders of magnitude. A clinical database must be capable of scaling elastically to handle this continuous growth, both in terms of raw storage capacity and query processing capacity, without service interruptions or disruptive architectural redesigns. This may involve implementing horizontal sharding, geographically distributed database architectures, or hybrid cloud solutions that allow for on-demand expansion of storage and computing resources without massive upfront capital investment.

For software developers working specifically with clinical databases, these ten characteristics translate into concrete and complex technical design decisions. The choice of the underlying database management system is crucial and often involves polyglot solutions: mature relational databases such as PostgreSQL, Oracle, or SQL Server remain the dominant choice for critical transactional structured data but are necessarily complemented by NoSQL databases such as MongoDB for semistructured documents, distributed object file systems for massive images and videos, and specialized timeseries databases such as InfluxDB for continuous physiological signals. The current architectural trend is unequivocally toward heterogeneous yet integrated

polyglot databases, where each data type is stored in the management system best suited to its access patterns, and a semantic integration layer provides the application layer with a unified and coherent view of the entire patient.

Data schema modeling is another critical aspect that requires deep medical expertise in addition to technical skills. The entity-relationship model of a clinical database must accurately reflect the complexity of the real medical domain: patients who have multiple episodes of care throughout their lives, each episode with multiple clinical observations of different types, each observation with multiple components and values, and complex relationships between entities that must be maintained with strict referential integrity. Unique patient identifiers, managing duplicates caused by registration errors, and resolving identities when the same patient appears with different data in different systems are complex algorithmic and organizational problems that require specific and often localized solutions. Security is a cross-cutting layer that affects all aspects of design and implementation. Access control must be granular down to the field level in some cases, allowing for the definition of which specific users can view which sensitive data, in which justified clinical contexts, and for what documented purposes. Encryption must be applied both in transit between system components and at rest on storage media, using robust algorithms and proper key management. Auditing must log all accesses and significant modifications with sufficient forensic granularity to enable subsequent investigations of incidents or claims. This entire security framework must be implemented without excessively degrading the performance experienced by clinical users and without unduly complicating the development of new features.

Disaster recovery is another critical aspect that cannot be left to chance or improvisation. Backups must be frequent, automatic, regularly verified, and stored in secure physical locations that are geographically separate from the main system to protect against natural disasters or major incidents. They must be periodically tested through recovery drills to ensure that they are truly recoverable within defined target times, not just theoretically. There must be a detailed business continuity plan specifying exactly how service will be restored in the event of a major catastrophic failure, with assigned roles and responsibilities, maximum recovery times contractually agreed upon with management, and procedures for communicating with stakeholders.

For clinicians and healthcare professionals who use these databases daily without consciously seeing them, understanding their fundamental characteristics helps them maintain realistic expectations about the system's behavior and collaborate constructively with technical teams. When a system takes time to respond to a complex query, it may legitimately be due to the semantic complexity of the search and not necessarily because the system is technically deficient. When historical data are not immediately available following a system migration, it may be due to necessary consistency and audit mechanisms, not an operational failure. When the system requests additional information or reauthentication to access certain sensitive data, it is for the protection of patient privacy and regulatory compliance, not due to unnecessary bureaucracy. This mutual understanding improves the collaborative relationship between clinical professionals and technical teams and reduces the mutual frustration that often arises from a lack of understanding of the other side's limitations and needs.

The future of clinical databases points toward an increasingly deep integration of diverse and massive data sources, including comprehensive genomic data from next-generation sequencing, continuous real-time data from connected devices, and social and environmental data that contextualize patient health. It also points toward the widespread use of artificial intelligence to automatically extract knowledge from data, with machine learning models trained directly on historical clinical databases to predict risks and suggest decisions. It points toward greater active involvement of patients themselves, who will be able to access their complete data, granularly control its secondary use, and contribute their own observations and data generated by wearables. These emerging developments pose new technical challenges of scalability and complexity, as well as new ethical dilemmas regarding privacy and consent, which the databases of the future will need to proactively address.

Clinical databases are not merely a technical repository of digital information but also the collective institutional and social memory of the healthcare system. They accumulate the knowledge generated by generations of medical and nursing professionals, the experiences of millions of patients who have entrusted their health to the system, and the lessons learned from countless individual clinical cases that have been meticulously documented. Preserving institutional memory, keeping it accessible for current and future clinical practice, and using it responsibly to improve care and research is an enormous responsibility shared by the developers who build these technical infrastructures, the administrators who operate them, and the clinicians who feed them with data and consult them for decision-making. Doing so well, with the highest

technical standards and the deepest ethical sensitivity, is among the most important and lasting contributions that can be made to people's health and the progress of medicine.

8.4 DESIGN FOR INTEROPERABILITY, NOT ISOLATION

"A perfect but isolated medical system is like a brilliant doctor who speaks only a language that no one else understands: its potential value is lost in the lack of communication."

Let us imagine that a software development team is tasked with building a new specialized application for the comprehensive management of patients with diabetes mellitus. They meticulously analyze the functional requirements with endocrinologists, design a technically elegant architecture, implement all the requested features with high-quality code, and deliver the final product within the agreed-upon timeline and budget. The hospital installs it on its infrastructure, the doctors begin using it with initial enthusiasm, and everything seems to work perfectly from an isolated technical perspective. A few months later, however, someone in quality assurance realizes that the diabetes application cannot automatically share structured data with the hospital's main electronic health record system, which operates as a standalone technological island. Doctors are forced to manually enter the same information twice: once in the specialized diabetes app for specific disease monitoring and once in the general medical record for a comprehensive view of the patient. Patient data remain fragmented across incompatible systems; automatic drug interaction alerts do not work because the diabetes app lacks visibility into the full range of medications the patient receives from other specialties, and the glycemic control reports generated by the app cannot be automatically incorporated into the patient's unified medical record for future reference. An app, technically perfect in its isolated design, is clinically an operational failure that generates frustration, transcription errors, and risks to patient safety. This story, which repeats itself with endless variations in hospitals around the world every year, painfully illustrates what inevitably happens when medical software is designed for isolation rather than explicitly designed for interoperability from the very first line of code.

Designing for interoperability means assuming from the very start of the project—even before writing the first technical requirement—that the system we are building will never exist in a technological vacuum but will necessarily be part of a broader ecosystem of other heterogeneous systems, with which it will need to exchange information in a fluid, bidirectional, and reliable manner to create real clinical value. It is not a superficial feature that can be added at the end of development, like a makeshift glue that binds systems that were not conceptually designed to understand one another. It is a fundamental property of software architecture, which must be present and actively considered in every design decision, from the choice of patient identification schemes to the definition of public APIs, including the internal data model and cross-cutting security mechanisms. Ignoring interoperability from the earliest stages of design is to condemn the product to live in isolation forever, and clinical users suffer the operational consequences of that lack of communication.

8.4.1 FUNDAMENTAL ARCHITECTURAL DECISIONS

The first design decision that critically and permanently affects the system's interoperability is the choice of identifiers for all relevant entities. Every patient, healthcare professional, clinical care episode, and data resource requires a unique identifier that can be shared, referenced, and resolved across different systems in an unambiguous manner. If each system uses its own internally generated identifiers without considering the ecosystem context, the subsequent mapping between them becomes an integration engineering nightmare that consumes resources and generates errors. Ideally, standardized national or international identifiers should be used whenever possible, such as the national medical record number, the licensed healthcare professional identifier, or episode codes following administrative standards. Alternatively, at the very least, an internal identification system should be designed that enables unambiguous exchange through identity resolution. Interoperable identifiers must be persistent over time (they do not change when the patient changes address or insurer), globally unique within the ecosystem's namespace (no two different patients have the same identifier), and resolvable via directory services (there must be a standard technical way to obtain further information from them).

The second critical architectural decision is the data model and the terminologies used to represent clinical information. A system consciously designed for interoperability cannot afford to invent its own idiosyncratic way of representing universal medical concepts, creating incompatible semantic silos. It must use, from the design phase, widely accepted and internationally validated terminological standards: HL7 FHIR for the structured exchange of clinical resources, SNOMED-CT for the semantically precise coding of clinical concepts, LOINC for the unambiguous identification of laboratory tests and observations, RxNorm or ATC for drug coding, and DICOM for medical image management. Using these standards from the outset of development ensures that other systems in the ecosystem can unambiguously understand the data our system produces and that we can correctly interpret the data we receive from external sources. Adapting a legacy system that used local proprietary terminologies to these international standards is technically possible but far more costly, complex, and prone to semantic mapping errors than incorporating them from the initial design.

The third decision is the controlled exposure of well-designed public APIs. A truly interoperable system is not an opaque black box; it is a system that exposes well-documented, stable, and versioned programming interfaces that allow other authorized systems to access its essential functionalities and data in a controlled manner. These APIs must follow modern web industry standards (RESTful architecture, JSON data formats, OAuth2 for authorization), be thoroughly documented with real-world use cases, and remain stable over time while adhering to semantic versioning rules that allow for evolution without breaking existing clients. They must be proactively designed with real integration use cases in mind that other systems will have, not just the immediate internal needs of the system being built. An API that exposes only the minimum the system needs to function internally—and not what other systems legitimately need to integrate with it—is an insufficient API that perpetuates isolation.

The fourth decision concerns the cross-system management of user identity and access control. Effective interoperability necessarily implies that authenticated users from other systems can access our data in an authorized manner and that our users can access data from other systems without having to memorize multiple credentials. This requires robust authentication and authorization mechanisms that operate in a federated manner across organizational and technological boundaries. Modern standards such as OAuth2 for delegated authorization and OpenID Connect for federated authentication, combined with healthcare-specific profiles such as SMART on FHIR, provide the complete technical framework to implement this securely and in a standardized manner. However, it must be designed and integrated from the very beginning of the security architecture, not as a retrofitted add-on, because it affects all layers of the system and all access control decisions.

The fifth decision concerns the flexible granularity of the exchanged data. A system designed for true interoperability must be capable of exchanging information at different levels of aggregation depending on the context of use and applicable privacy restrictions. Sometimes the most granular data possible are needed: the individual point-in-time value of a capillary blood glucose reading with its exact timestamp. Other times, an aggregated statistical summary suffices: the average blood glucose levels from the past week or the percentage of values within the target range. The design of the API and the data model must explicitly allow for both query possibilities and must include descriptive metadata that clearly indicate the level of aggregation applied and the context in which the data were generated. This is especially important for the protection of privacy: sometimes it is necessary to share aggregated population data for research or planning without revealing identifiable information about specific patients, and the system must be able to generate and serve these aggregates reliably.

The sixth decision concerns the explicit management of the synchronization and consistency of distributed data. When multiple heterogeneous systems share and replicate information about the same patients, the complex technical problem of maintaining consistency over time inevitably arises. If one system updates a clinical data point, how do the other systems maintaining related copies determine? What happens if two different systems update the same data point simultaneously with different values? Who holds the authority for the definitive “truth” for each data type? There are no universal answers valid for all cases; they depend critically on the clinical context and the specific use case. However, mechanisms must be consciously designed to address these questions: event systems that notify interested subscribers of relevant changes, APIs that allow explicit retrieval of the most recent available version, and documented conflict resolution policies when inconsistencies occur. Eventual consistency is often sufficient and acceptable for many noncritical clinical use cases but is not always acceptable, especially for those involving immediate patient safety.

The seventh decision concerns the complete traceability of data provenance and transformation. When data travel between multiple systems in an interoperable ecosystem, it is essential to maintain an immutable record of its original source and all the transformations it undergoes along the way. Does these laboratory result data come directly from the laboratory's automated analyzer, or was it manually entered by a technician? Has it been technically validated and signed off by a responsible clinician? Has it been subsequently modified by any normalization or correction algorithm? Complete traceability allows clinicians to critically evaluate the reliability and context of the information before making medical decisions on the basis of it. It is also absolutely essential for regulatory audits and for forensic investigations of security or quality incidents. Designing for traceability involves including metadata on provenance, authorship, and transformation in all data exchanges, not just in the data at rest.

The eighth decision involves robust and transparent management of communication errors. In an interoperable distributed system, communication failures are statistically inevitable and must be anticipated architecturally. The network may fail intermittently, an external system may be temporarily down for maintenance, or a message may become corrupted in transit because of line noise. The system design must include explicit and robust error-handling mechanisms: automatic retries with exponential backoff to avoid saturation, persistent message queues that survive system reboots, clear notifications to human operators when nonrecoverable failures are detected, and above all, the ability for end-user clinicians to explicitly know when the information they are viewing on screen may be out of date, incomplete, or potentially inconsistent because of synchronization issues. Operational transparency regarding the actual state of interoperability is essential for maintaining clinical trust in the system.

8.4.2 DEVELOPMENT CULTURE AND SHARED RESPONSIBILITY

For software developers working in the healthcare sector, explicitly designing for interoperability requires a profound shift in mindset compared with other, less-regulated industries. It is no longer just about building a system that functions correctly in isolation but about building a system that functions well as a cooperative member of a broader and more complex ecosystem. This requires investing time in deeply understanding industry standards, actively participating in professional communities where these standards are discussed and evolved, and remaining abreast of regulatory and technological developments. It also requires a certain degree of technical humility: recognizing that our system, however brilliant it may be, is not the center of the hospital's technological universe but rather just another node in a complex network of interdependent systems, each with its own history, limitations, and value. In addition, professional generosity is needed: thoroughly documenting APIs and data models to make life easier for other developers who will need to integrate with us, offering proactive technical support to integration teams, and contributing constructively to the continuous improvement of shared standards through feedback on real-world experience.

For clinicians and healthcare managers involved in information system procurement or development projects, designing for interoperability means making it a nonnegotiable requirement that the systems purchased or commissioned explicitly comply with recognized interoperability standards. It means systematically asking, before signing any procurement contract: Can this system automatically share structured data with our primary electronic health records? What specific terminology and exchange standards does it use? How is the unique patient identity managed across systems? Does it offer documented and stable APIs for integration? This means never settling for technically brilliant solutions that operate in isolation, no matter how good they may be in their specific functional domain, because an isolated solution is not truly a solution for the patient; it is a future problem of information fragmentation and security risk.

The history of health informatics is unfortunately full of examples of systems that were consciously designed for isolation—sometimes due to a vendor-lock-in business strategy, sometimes due to ignorance of the implications, and sometimes due to a rush to deliver. Clinical laboratory systems that generate results that cannot automatically flow into the electronic health record. Hospital pharmacy systems that lack real-time visibility of allergies and contraindications are recorded in the emergency department. Chronic patient monitoring applications that generate valuable remote monitoring data that no one else on the care team can view or use. Each of these systems, considered individually within its silo, may be technically excellent and fully meet the functional requirements for which it was designed. However, together, in the real patient care ecosystem, they form an archipelago of isolated technological islands where information is

critically lost, dangerously duplicated, and silently corrupted, and those who pay the ultimate price are the patients who receive fragmented, discontinuous, and potentially unsafe care.

Designing for interoperability is neither technically free nor trivial. It requires investing additional time in the initial design, sustained effort during implementation, and maintaining up-to-date specialized knowledge within the team. However, it is an investment with a guaranteed long-term return. An interoperable system can integrate seamlessly into the hospital's existing technological ecosystem; it can be expanded with new functionalities in the future without disrupting what already works, and it can eventually be replaced by a better system without losing accumulated data or operations. An isolated system, on the other hand, is a long-term trap: once it is adopted and filled with critical data, it is very difficult and costly to move away from it without losing valuable information or undergoing traumatic migration that paralyzes clinical activity.

The global regulatory trend in digital health is unequivocally toward a growing demand for interoperability as a market requirement. In the United States, the ONC (Office of the National Coordinator for Health Information Technology) information rule now requires certified electronic health record providers to expose standard FHIR APIs for data access. In Europe, the European Health Data Space is aggressively driving the adoption of common standards for cross-border interoperability. In many countries, public procurement of healthcare systems already explicitly requires compliance with proven interoperability profiles as a bidding requirement. Designing for interoperability is not just an optional technical best practice; it is a growing regulatory requirement for legal competition in the digital health market.

Once we have designed systems that can interoperate and share information seamlessly, how do we protect all that sensitive information from the growing cybersecurity threats that specifically target the healthcare sector, and how do we balance the openness necessary for interoperability with the protection essential for privacy?

REFERENCES

Mell P, Grance T. The NIST Definition of Cloud Computing. September 28, 2011 [cited May 13, 2026]; Available from: <https://csrc.nist.gov/pubs/sp/800/145/final>
What Is Edge Computing? | IBM [Internet]. [cited May 13, 2026]. Available from: <https://www.ibm.com/think/topics/edge-computing>