IJC International Journal of Computer
Peer-Reviewed International Journal
ISSN: 2307-4523
https://ijcjournal.org

A Typology of Phased Migration Strategies for Legacy JSP Interfaces of Clinical Information Systems to React Component Architecture

Md Thofiq Imamsab*
Software Engineer, United States
* Corresponding author: developerthofiq@gmail.com

Accepted: 28 September 2026 | Published: 24 October 2026

Abstract
This article investigates the architectural conditions under which legacy JavaServer Pages (JSP) interfaces in clinical information systems can be migrated to React while controlling risks to workflow continuity and data consistency. A structured integrative review combined with qualitative configurational synthesis was conducted using 15 publications selected from an initial set of 43 sources. Quantitative evidence was extracted to identify constraints associated with concurrent use, functional coupling, device integration, migration complexity, and clinically significant interface behavior. The analysis shows that migration scope alone is an insufficient basis for strategy selection. Shared mutable state, cross-module dependencies, and the consequences of regression determine whether a component, route, workflow, or subsystem can be transferred safely. Four archetypal migration configurations were derived by aligning the functional boundary with the coexistence mechanism, backend state authority, regression assurance, and recovery capability. The resulting Linked-Decision Model treats clinical criticality as a constraint on the minimum assurance level and distinguishes code rollback from the recovery or reconciliation of persisted data. A retrospective laboratory-scheduling case was used as an illustrative consistency check rather than empirical validation. The model provides a structured basis for deciding when phased coexistence is appropriate and when controlled full cutover may present lower risk.

Keywords: healthcare software; system modernization; clinical workflows; data consistency; software testing; risk management; patient safety

1. Introduction

Legacy JavaServer Pages (JSP) interfaces remain in clinical systems due to accumulated application logic, long service life, and numerous internal dependencies [1]. In such applications, server-side page generation often couples the presentation with controllers, the session, and domain objects, so user actions may be accompanied by a full page reload, while interface changes may affect the server side [2]. In the clinical process, this coupling increases the cost of regression. An error can disrupt work with a laboratory order, schedule, or examination result [3]. Component-based frontend architectures support the gradual extraction of functions from a Java/JSP monolith [2]. React is examined in this study as the target implementation of this broader architectural approach.

A complete rewrite changes all pages and user scenarios within a single release boundary. Page-level replacement may preserve dependencies that span an end-to-end workflow, whereas component embedding retains direct interaction between legacy and new code. Microfrontends isolate individual functional areas [2]. Their application requires management of architectural boundaries and dependencies [4]. Parallel operation of subsystems creates a risk of data divergence [5]. A comparison of the reviewed studies revealed a gap in the classification of the transition from JSP to React. Existing approaches consider the unit of

migration, coexistence, state management, and regression control separately. An integrated model should also take into account clinical criticality, which determines the acceptable scope of failure and rollback requirements. Migration granularity constrains the feasible coexistence boundary. This boundary influences state authority, which in turn determines the minimum required level of regression assurance.

The aim of the study is to develop a model for selecting a phased strategy for migrating legacy JSP interfaces in clinical information systems to a React component architecture, considering the migration unit, coexistence boundary, state authority, and regression assurance. To achieve this aim, the following tasks were set:

– identify the architectural parameters by which phased migration strategies differ;
– systematize the quantitative characteristics of clinical information systems presented in the literature that affect the architectural constraints of migration;
– compare alternative migration units, coexistence boundaries, state-authority models, and levels of regression assurance;
– identify stable combinations of the four characteristics and form a typology of strategies based on them;
– determine the applicability conditions of each strategy, taking into account clinical criticality and rollback requirements;
– evaluate the internal consistency of the proposed typology through a retrospective professional account of laboratory-scheduling interface migration.

The research hypothesis is that the safety and applicability of phased migration depend on the alignment of the migration unit, coexistence boundary, state authority, and regression assurance. A small migration unit permits a narrow coexistence boundary and limited assurance only when client-side state is ephemeral and significant cross-module dependencies are absent. Workflow- and subsystem-level migration require stronger backend authority, broader end-to-end assurance, and explicit provisions for code rollback and persisted-data recovery.

The scientific contribution of the study is not the introduction of individual migration mechanisms such as component extraction, route strangulation, or microfrontend decomposition, which have been discussed in previous research. The contribution lies in integrating four decision dimensions—migration unit, coexistence boundary, state authority, and regression assurance—into a single rule-based framework for clinical interface migration. The model additionally introduces clinical criticality and persisted-data recovery as constraints on the applicability of a migration configuration. This integration distinguishes architectural decomposition from the assurance and recovery requirements needed for safe coexistence and provides a basis for selecting a migration boundary according to the structure of clinical workflows rather than according to frontend technology alone.

2. Materials and Methods

The study combined a structured integrative review, descriptive quantitative extraction, and qualitative configurational synthesis. The synthesis was used to construct a model for selecting phased strategies for migrating legacy JSP interfaces in clinical information systems to a React component architecture. The

evidence base covered legacy-system modernization, electronic health record architecture, modular frontend design, microfrontends, and phased migration methods. Studies on shared state management, organization of interaction between old and new architectures, contract and end-to-end testing, regression control, and rollback of changes were analyzed.

The literature search was conducted through Google Scholar, PubMed, SpringerLink, IEEE Xplore, ACM Digital Library, MDPI, and arXiv. The final search was completed in August 2026, and the eligibility window covered publications issued between January 2022 and December 2025. The following queries and their combinations were used: “legacy system modernization,” “JSP migration,” “JSP to React migration,” “frontend modernization,” “React migration,” “microfrontend migration,” “strangler pattern,” “EHR architecture,” “clinical information system,” “health information system migration,” “state management,” “Server-Sent Events,” “contract testing,” “Pact testing,” “end-to-end testing,” and “regression testing.” The AND and OR operators were used to refine the search. The sample included publications from 2022–2025 containing analytical, empirical, or methodological information on the modernization of legacy systems, the architecture of clinical applications, modular interfaces, and migration risk control. Studies devoted exclusively to server-side technologies without conclusions applicable to interface migration, as well as materials without a description of architectural solutions or a research basis, were excluded.

The search identified 43 records. Titles, abstracts, and available full texts were screened against the eligibility criteria. Fifteen publications were retained, while 28 records were excluded because they lacked a relevant architectural contribution, sufficient methodological description, or findings transferable to phased interface migration. The analysis showed that the literature examines in detail the migration of monolithic systems, the architecture of medical information systems, and individual approaches to the modular construction of interfaces. At the same time, the information is distributed unevenly: issues of server-side modernization have been studied significantly more deeply than the transition from JSP to React, while state management, the coexistence of two architectures, and regression control are more often considered as independent tasks.

The configurational synthesis used four axes: migration unit, coexistence boundary, state authority, and regression assurance. Quantitative and qualitative evidence relevant to these axes was extracted where available, including numbers of users, modules, devices requiring integration, migration activities, evaluation participants, architectural risks, and documented antipatterns. Configurations were screened using five compatibility rules: clinical domain state required backend authority; the coexistence boundary had to contain the migrated function; assurance had to cover every write path affecting authoritative state; production adoption required feasible code rollback and persisted-data recovery; and technology-specific variants with equivalent risk profiles were consolidated. This procedure produced four architectural archetypes rather than empirical clusters. The retrospective laboratory-scheduling account was used solely as an illustrative consistency check.

The main limitation of the study is related to the heterogeneity of the sources. The publications differ in subject area, research design, scale of the systems under consideration, and completeness of quantitative data. Some studies are devoted to server-side migration or microfrontends outside healthcare, so their results were used only to analyze transferable architectural principles. The retrospective professional account was reconstructed without preserved event logs or project documentation. It was therefore used as an illustrative

consistency check rather than as empirical validation of the model. The selected body of sources provides a conceptual basis for the typology, but its statistical and cross-project validation requires the analysis of several independent clinical systems.

3. Results

The descriptive extraction yielded five constraint categories relevant to phased interface migration: concurrent use, functional scope, device integration, migration complexity, and clinical consequences of interface design. The reported values were interpreted as contextual indicators rather than universal thresholds. Their architectural significance depends on the relationships among users, data, modules, and external systems.

In a single hospital information system, 6,165 users were active and an average of 800, or 13%, worked concurrently [5]. This volume indicates substantial simultaneous activity but does not independently justify backend-authoritative state or a particular migration strategy. Stronger state control becomes necessary when several users can modify the same schedule, order, or other workflow object. Where concurrent users access independent records, the primary implication concerns capacity planning rather than state authority.

The same system contained almost 50 modules, including 19 major modules. Only 6 of the 19 major modules exceeded 70% functional fit with the replacement system [5]. These values describe functional scale and heterogeneity, but module count does not establish releasability. A module can serve as an independent migration unit only when its data dependencies, server-side rules, and cross-module operations are sufficiently bounded.

The integration inventory comprised 112 devices requiring integration: 81 laboratory devices and 31 other medical devices [5]. This value must not be interpreted as the number of technical interfaces or data flows. It represents devices whose continued operation depended on integration with the new system. Such a boundary requires compatible APIs, controlled ownership of records, and verification of data transmission. Where these conditions cannot be maintained during coexistence, a planned full cutover may present less risk than parallel operation.

Migration research based on 19 interviews and 215 developer discussions distinguished two modes of change, 14 migration activities, and 53 solution outcomes [6]. Twenty outcomes concerned system-level transformations and 33 concerned technical transformations. These findings show that interface migration cannot be reduced to source-code replacement. Release organization, service boundaries, operational monitoring, and deployment processes must evolve with the migrated interface.

The clinical evidence base comprised 32 studies of electronic health record design. User satisfaction was assessed in 21 studies, effectiveness in 10, efficiency in 9, and medication safety in 6, with overlap among categories [3]. Seven design themes were identified: searchability, automation, customization, data entry, workflow, user guidance, and interoperability. For a clinically significant function, regression assurance must therefore extend beyond technical rendering and include information availability, data-entry behavior,

workflow continuity, and system interaction. The quantitative evidence and the implications derived for the proposed typology are separated in Table 1.

Table 1: Quantitatively supported clinical and architectural constraints and their implications for phased JSP-to-React migration [3], [5], [6]

Migration constraint	Evidence reported in the source	Architectural constraint derived in this study	Implication for the proposed typology
Concurrent use	6,165 active users; average of 800 concurrent users (13%)	Concurrent load requires stronger state control when users modify shared workflow objects	Type III–IV for shared mutable workflows; capacity scaling for independent records
Functional scope	Almost 50 modules; 19 major modules; 6 of 19 major modules exceeded 70% functional fit	Module count does not establish releasability; dependencies and functional fit must be mapped	Type II for autonomous routes; Type III–IV for coupled workflows or subsystems
Integration surface	112 devices requiring integration: 81 laboratory devices and 31 other medical devices	Device continuity requires stable interfaces, backward compatibility, and controlled data exchange	Type IV where safe coexistence and reconciliation are feasible; controlled full cutover in other configurations
Migration complexity	19 interviews; 215 Stack Overflow discussions; 2 modes of change; 14 migration activities; 53 solution outcomes	Technical and systemic changes must be coordinated across migration iterations	Supports staged planning across Types II–IV without directly favoring one type
Clinical interface effects	32 studies; 21 assessed satisfaction, 10 effectiveness, 9 efficiency, and 6 medication safety; 7 design themes	Regression assurance must cover usability, workflow, data entry, guidance, and interoperability	Clinical criticality sets the minimum assurance level across all four types

The extracted values delimit the decision problem but do not select React or determine a migration type by themselves. The relevant condition is the relationship between scale and coupling. Eight hundred concurrent users become significant for state authority when they can modify a shared object. Nineteen major modules become significant for migration granularity when their rules and data cross module boundaries. Devices requiring integration expand the verification boundary only to the extent that their data flows interact with the migrated subsystem.

The coding framework assigned four possible values to each of four axes: migration unit, coexistence boundary, state authority, and regression assurance. The resulting theoretical configurations were not treated as observed cases or empirical clusters. They represented possible architectural arrangements subjected to rule-based compatibility screening.

The screening rules were derived from the requirement that the functional, state, testing, and recovery boundaries remain aligned. Architecture-centric migration research supports the use of explicit decision criteria rather than technology-led decomposition [7]. Contemporary modernization research also shows that approaches tied to a single technology or scenario provide limited guidance outside their original setting [8]. The rules were therefore formulated independently of a particular React library, state-management package, bundler, or event transport. The compatibility rules are presented in Table 2.

Table 2: Compatibility rules used to derive the archetypal migration configurations [1], [6], [7]

Rule	Compatibility test	Incompatible or merged configuration	Consequence for the typology
R1. Domain-state authority	Clinical domain data must have a server-side authoritative source; browser state may be authoritative only for temporary presentation data	A component or route holding the final version of an order, result, or shared schedule in the browser is excluded	Type I permits local ephemeral UI state; Types II–IV retain backend authority over domain state
R2. Boundary completeness	The coexistence boundary must fully contain the migrated function	A page-level boundary is excluded when the clinical operation depends on write paths located on other routes	The configuration is reassigned from a route to a workflow or subsystem boundary
R3. Assurance coverage	Regression assurance must cover every component or system capable of changing authoritative state	Component-only testing is excluded for shared workflows; route-only testing is excluded for parallel subsystems	Assurance expands from integration to contracts, workflow-level E2E testing, and reconciliation
R4. Rollback and recovery feasibility	Code rollback requires compatible APIs; persisted operations require verification, reconciliation, or recovery	A traffic switch without a procedure for handling data written through the new implementation is excluded from production use	Strategy adoption depends on both code rollback and data recovery capability
R5. Configurational equivalence	Configurations with the same functional boundary, state authority, minimum assurance, and recovery requirement are considered equivalent	Variants differing only in library, bundler, event transport, or deployment tooling are merged	Technology-specific variants are consolidated into four architectural archetypes

Application of R1 excluded configurations in which a browser held the authoritative version of clinical domain data. R2 reassigned nominally component- or page-level migrations when their functional dependencies crossed the selected boundary. R3 eliminated testing strategies narrower than the state-changing surface. R4 separated deployable configurations from those lacking a feasible rollback and recovery path. R5 then merged variants whose architectural risk profile remained unchanged despite differences in implementation technology.

This procedure produced four archetypal configurations organized by their dominant functional boundary: component, route, workflow, and subsystem. The four types are analytical reference configurations rather than an empirical ranking. Type IV is not more mature than Type I; it addresses a broader state and integration boundary.

Type I – Embedded Component Migration. A React component is mounted within a JSP page while the backend remains authoritative for clinical domain data. Client-side state is limited to temporary presentation concerns such as filters, panel visibility, or an unsaved selection. Regression assurance covers component behavior and direct integration with the surrounding page. The type is applicable when the function has limited dependencies and can be returned to its JSP implementation while preserving API compatibility and the validity of operations already persisted through React.

Type II – Route Strangler Migration. An autonomous page or route is migrated while the remaining routes continue to use JSP. A stable application programming interface (API) separates the React screen from backend implementation details. Integration and contract tests verify access control, request and response compatibility, and persistence behavior. The route remains an appropriate migration unit only while its operations form a bounded domain context.

Type III – Workflow Microfrontend Migration. The migration unit comprises a connected clinical workflow that may span several screens and users. An isolated React microfrontend coexists with JSP, while

the backend stores shared authoritative state. Event propagation distributes confirmed changes, whereas ordering, version control, conflict handling, and persistence remain backend responsibilities. Contract tests are combined with end-to-end (E2E) and concurrent-user scenarios.

Type IV – Parallel Subsystem Migration. The boundary covers a clinical subsystem together with its workflows and external dependencies. The legacy and replacement implementations operate in parallel under a single-writer rule or a formally defined reconciliation process. Regression assurance extends to cross-system E2E testing, data comparison, switching, and recovery. The configuration is acceptable only when coexistence does not create uncontrolled write paths or irreconcilable records. Application of the compatibility rules produced four archetypal configurations. Their migration boundaries, state models, assurance requirements, and recovery conditions are compared in Table 3.

Table 3: Author-developed typology of phased migration strategies for legacy JSP clinical interfaces. [1], [2], [5]

Strategy	Migration unit	JSP/React coexistence	State authority	Minimum regression assurance	Code rollback and data recovery requirement	Acceptance condition
Type I – Embedded Component	User interface component	React embedded in a JSP page	Local ephemeral UI state; backend-authoritative domain data	Component and integration testing	Feature toggle or JSP fallback; review of any operations submitted before rollback	Isolated function with limited dependencies and assurance matched to clinical criticality
Type II – Route Strangler	Page or route	JSP and React routes coexist	Backend state mediated through stable APIs	Integration and contract testing	Route switch to JSP with API compatibility; verification of data saved through the React route	Autonomous screen with bounded cross-route dependencies
Type III – Workflow Microfrontend	Clinical workflow	Isolated React microfrontend coexists with JSP	Backend-authoritative shared state with event propagation	Contract, multi-user, and workflow-level E2E testing	Workflow traffic switch where supported; verification or recovery of persisted clinical operations	Shared mutable state, concurrent actors, and a workflow boundary suitable for independent release
Type IV – Parallel Subsystem	Clinical subsystem	Legacy and replacement subsystems operate in parallel	Single-writer backend authority with reconciliation	Cross-system E2E and reconciliation testing	Subsystem switch combined with a formal data reconciliation and recovery procedure	High clinical criticality, broad integration surface, and feasible reconciliation

The retrospective professional account maps to Type III. Laboratory scheduling formed the functional migration boundary, and an isolated React microfrontend operated alongside JSP. Concurrent changes to a shared schedule led to backend-authoritative state, while Server-Sent Events (SSE) propagated confirmed updates. Pact tests checked API contracts, and E2E scenarios covered the path from laboratory order through scheduling to result archival.

The retrospective account estimated that response time decreased from approximately 2.5 seconds to 300 milliseconds and that interface delays consumed 3–4 coordinator hours per day. Reports associated with delayed updates were recalled as decreasing from approximately 8–12 per week to almost zero after state authority moved to the backend. These values were reconstructed from memory and were excluded from Table 1 because no event logs, support records, or measurement protocol were available.

The account illustrates the internal coherence of the proposed Type III configuration. It does not provide empirical validation, establish comparative effectiveness, or document the implementation of workflow-level rollback. The evidence is consistent with the research hypothesis only in its conditional form: broader

assurance and stronger backend authority become necessary when expansion of the migration boundary is accompanied by shared mutable state or external integration dependencies.

4. Discussion

The typology explains why phased migration is often preferable to a complete rewrite without treating incremental change as a universal rule. Migration research remains fragmented across technical and organizational concerns, while many approaches concentrate on individual transformation activities [1]. A mapping of 43 studies identified six modernization macro-activities, confirming that migration extends beyond code conversion [9]. Architecture-centric guidance similarly treats modernization as a sequence of decisions rather than a framework replacement operation [7].

In a clinical interface, phased migration limits the scope in which a defect can affect an operational workflow. A component or route can be introduced while unaffected JSP functionality remains available. This containment is valuable because electronic health record (EHR) interface design influences information retrieval, data entry, workflow execution, and medication safety [3]. Coexistence therefore acts as a risk-control mechanism when the migrated boundary, authoritative state, and rollback path are explicit.

The same coexistence can become a source of risk. Parallel implementations may duplicate write paths, require repeated synchronization, or leave persisted records in different states. The hospital migration case selected a non-incremental go-live because the scale of system and device integration made prolonged coexistence difficult to control [5]. A complete cutover is consequently justified when a single transition can be tested comprehensively, the legacy system can remain available for read-only reference, and parallel writes would create greater uncertainty than the cutover itself.

Types I and II are preferable where functions can be isolated without splitting a clinical workflow. Type III is warranted when shared state or concurrent actors make a page boundary artificial. Type IV applies only where single-writer authority and reconciliation make parallel operation credible. The model therefore selects a risk-matched boundary rather than automatically favoring the smallest migration unit.

The four archetypes describe migration from server-rendered interfaces to component-based frontends at an architectural level. JSP and React constitute the implementation examined in the professional account, but the compatibility rules are not unique to these technologies. This distinction defines the limits of the claimed contribution: the typology concerns migration boundaries, state authority, assurance, and recovery, while React supplies the target component model.

The Java/JSP case described by Kojo et al. [2] supports the feasibility of gradually extracting pages into vertical microfrontends. Its replacement stack used Svelte and Fastify rather than React, so it cannot serve as direct evidence for React-specific migration. It does, however, support route- and domain-oriented strangulation of a JSP monolith. The same study found that half of eight participants did not recognize the resulting architecture as a microfrontend. This result indicates that vertically sliced microfrontends require explicit architectural documentation and shared terminology; it does not establish a general requirement for operational observability [2].

React-specific evidence in the reviewed corpus is narrower. Modular rendering and adaptive hydration show how React interfaces can be divided into independently rendered and hydrated units [10]. That work remains conceptual and leaves empirical evaluation for future research. It supports the technical feasibility of modular React composition but does not demonstrate the effectiveness of JSP-to-React migration. The proposed typology must therefore be understood as a technology-independent architectural model instantiated for JSP and React, rather than as a classification empirically derived from React migration projects.

Microfrontend tooling cannot substitute for explicit boundaries. A catalog of 12 anti-patterns evaluated by 20 practitioners identified recurrent problems involving decomposition, dependencies, operations, and team decisions [4]. Typed module federation may strengthen module contracts, but debugging, observability, and performance remain open implementation concerns [11]. These findings support governance of Type III boundaries without making a particular federation mechanism part of the type definition.

Clinical interfaces participate in broader information structures than the screen visible to the user. An integrated personal health record architecture derived requirements from health organizations and patients and separated business, application, data, and technology concerns [12]. This separation shows why a migration boundary defined only by page layout may fail to capture the actual clinical process.

The same issue appears in EHR-integrated data capture. One implementation connected 19 electronic data capture systems serving 2,217 patients across 11 clinical departments [13]. At this scale, interface changes must preserve data provenance and exchange across systems that outlive an individual frontend. Reviews of open-source EHR systems likewise identify modularity as useful but report continuing limitations in standards compliance and implementation maturity [14]. A modular interface therefore does not remove the need for stable domain and interoperability boundaries.

Service-oriented and decentralized patterns in health information systems increasingly rely on formalized contracts, including Fast Healthcare Interoperability Resources (FHIR), to stabilize exchange boundaries [15]. For the proposed typology, this evidence supports the distinction between visual decomposition and state authority. A React component may own its presentation behavior, but the API and backend retain authority over clinical domain state. Where several systems can modify that state, the migration unit must expand until every write path is governed by a common contract or reconciliation rule.

The central relationship is conditional rather than strictly monotonic. Increasing migration granularity generally broadens the coexistence and regression surfaces. Stronger backend authority and broader assurance become necessary when that expansion also introduces shared mutable state, concurrent actors, or cross-system dependencies. A visually small component can still require workflow-level assurance if it modifies a critical order, while a large read-only route may remain compatible with a narrower testing boundary.

The decision sequence is presented in Figure 1. The clinical workflow defines the functional boundary, which constrains the feasible migration unit and coexistence mechanism. All write points determine state authority, while the state and integration surfaces determine minimum regression assurance. Clinical criticality can raise that minimum, and rollback capability governs whether the configuration may be introduced into production.

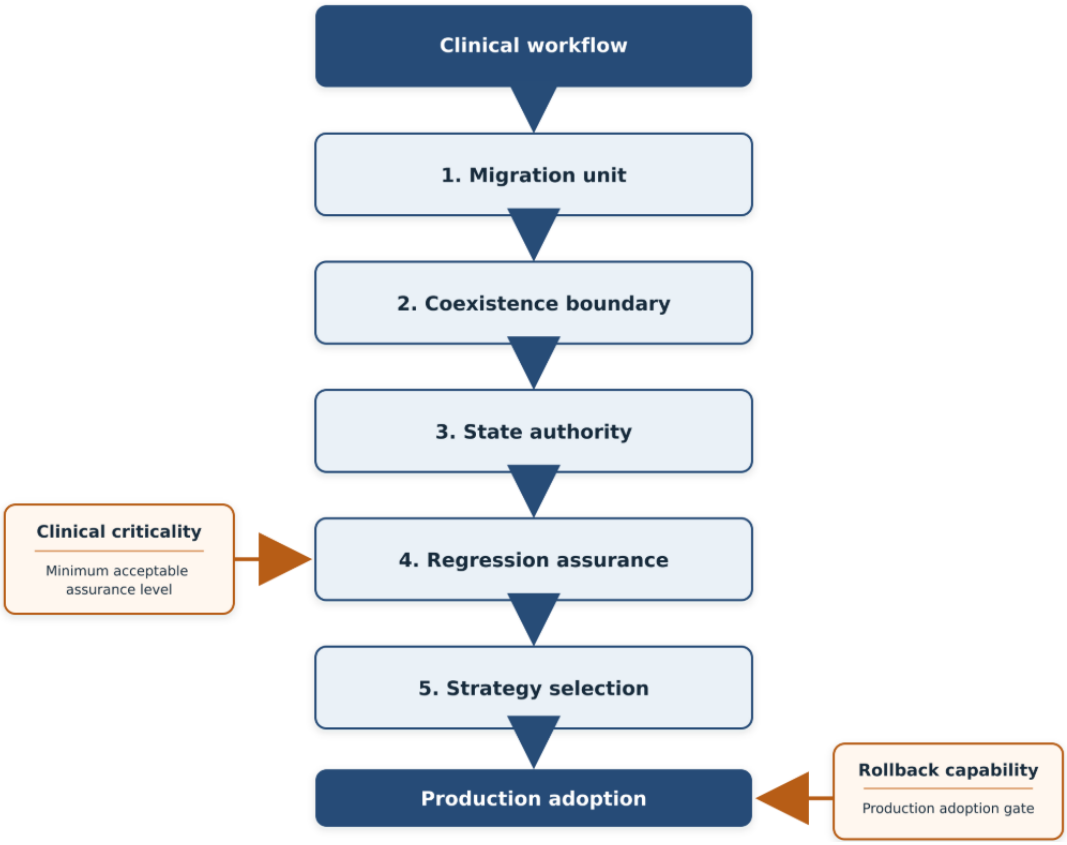


Figure 1: Linked-Decision Model for Clinical Interface Migration: migration unit → coexistence boundary → state authority → regression assurance → strategy selection (Author's own development)

The model rejects a configuration when its boundaries do not align. A component-local state model is unacceptable for shared clinical data. Route-level assurance is insufficient when operations cross several screens. Parallel subsystem operation is unacceptable without a single writer or reliable reconciliation. Code rollback alone is also insufficient when the new implementation has already created or modified persistent clinical records.

The model is intended to guide architectural reasoning rather than calculate an optimal strategy. It does not assign universal thresholds for users, modules, devices, or clinical criticality. Modernization approaches frequently remain tied to a specific technology or organizational setting, which limits direct transfer between projects [8]. Local dependency mapping, workflow analysis, and recovery requirements must therefore precede selection of an archetype.

The evidence base combines healthcare architecture studies, legacy modernization research, microfrontend cases, and conceptual frontend work. These sources differ in design, scale, and maturity.

Several examine backend migration rather than interface migration, while some microfrontend studies are based on nonclinical systems or remain preprints. Their findings support architectural principles but do not provide comparative estimates of migration effectiveness.

The four archetypes were produced through conceptual compatibility screening rather than statistical clustering. The screening and consolidation procedure has not yet been independently replicated, and inter-rater agreement was not assessed. Future research should publish the extraction matrix, apply the compatibility rules to multiple clinical projects, and evaluate agreement between independent reviewers.

The professional account provides an illustrative consistency check based on a single project and lacks primary telemetry and retained project documentation. Comparative validation requires independent JSP-to-React migrations measured using common indicators: migration duration, production defect leakage, regression effort, frequency of code rollback, data-recovery incidents, and deployment impact. Until such evidence is available, the typology should be treated as a structured decision model whose internal logic is supported by the reviewed evidence, not as a proven hierarchy of migration strategies.

5. Conclusion

The suitability of a phased JSP-to-React migration depends on the alignment of the functional boundary, coexistence mechanism, state authority, and regression assurance. React provides the target component model but does not determine migration safety. A small interface element may still require workflow-level assurance when it modifies shared clinical data, while a larger read-only unit may remain compatible with narrower controls.

Coexistence reduces migration risk only when the backend retains authority over clinical domain state, testing covers every write path, and rollback addresses both code and persisted data. Where these conditions cannot be maintained, controlled full cutover may be safer than parallel operation.

The proposed typology offers a structured basis for architectural selection rather than a hierarchy of strategies. Its broader validity requires evaluation across independent clinical systems using comparable measures of defect leakage, regression effort, rollback frequency, data-recovery incidents, and deployment impact. The findings provide conceptual support for the research hypothesis that safe phased migration requires alignment between migration unit, coexistence boundary, state authority, and regression assurance. However, the hypothesis cannot be considered empirically confirmed because the proposed typology was constructed through rule-based synthesis and examined using only one retrospective professional account without independent project telemetry or comparative validation.

References

[1] K. Tuusjärvi, J. Kasurinen, and S. Hyrynsalmi, “Migrating a legacy system to a microservice architecture,” e-Informatica Software Engineering Journal, vol. 18, no. 1, Art. no. 240104, 2024, doi: 10.37190/e-Inf240104.

[2] R. H. H. Kojo, L. F. C. Real, R. C. Ferreira, T. de O. Rosa, and A. Goldman, “Exploring micro frontends: A case study application in e-commerce,” arXiv preprint, 2025, doi: 10.48550/arXiv.2506.21297.

[3] M. Cahill, B. J. Cleary, and S. Cullinan, "The influence of electronic health record design on usability and medication safety: systematic review," BMC Health Services Research, vol. 25, Art. no. 31, 2025, doi: 10.1186/s12913-024-12060-2.

[4] N. P. S. da Silva, E. Rodrigues, and T. Conte, "A catalog of micro frontends anti-patterns," arXiv preprint, 2024, doi: 10.48550/arXiv.2411.19472.

[5] M. Kalkatawi, "Beyond the upgrade: unraveling the complexities of health information system migration," Discover Health Systems, vol. 4, Art. no. 7, 2025, doi: 10.1007/s44250-025-00186-x.

[6] H. Michael Ayas, P. Leitner, and R. Hebig, "An empirical study of the systemic and technical migration towards microservices," Empirical Software Engineering, vol. 28, Art. no. 85, 2023, doi: 10.1007/s10664-023-10308-9.

[7] J. Fritzsch, J. Bogner, M. Haug, S. Wagner, and A. Zimmermann, "Towards an architecture-centric methodology for migrating to microservices," in Agile Processes in Software Engineering and Extreme Programming – Workshops, P. Kruchten and P. Gregory, Eds. Cham, Switzerland: Springer, 2024, pp. 39–47, doi: 10.1007/978-3-031-48550-3_5.

[8] W. K. G. Assunção, L. Marchezan, A. Egyed, and R. Ramler, "Contemporary software modernization: Perspectives and challenges to deal with legacy systems," arXiv preprint, 2024, doi: 10.48550/arXiv.2407.04017.

[9] L. F. Fávero, N. R. de Almeida, and F. J. Affonso, "A systematic mapping study on the modernization of legacy systems to microservice architecture," Applied System Innovation, vol. 8, no. 4, Art. no. 86, 2025, doi: 10.3390/asi8040086.

[10] K. Chen, "Improving front-end performance through modular rendering and adaptive hydration (MRAH) in React applications," arXiv preprint, 2025, doi: 10.48550/arXiv.2504.03884.

[11] B. Lando and W. Hasselbring, "Toward bundler-independent module federations: Enabling typed micro-frontend architectures," arXiv preprint, 2025, doi: 10.48550/arXiv.2501.18225.

[12] N. C. Harahap, P. W. Handayani, and A. N. Hidayanto, "Integrated personal health record in Indonesia: Design science research study," JMIR Medical Informatics, vol. 11, Art. no. e44784, 2023, doi: 10.2196/44784.

[13] M. Dugas et al., "Next-generation study databases require FAIR, EHR-integrated, and scalable Electronic Data Capture for medical documentation and decision support," npj Digital Medicine, vol. 7, Art. no. 10, 2024, doi: 10.1038/s41746-023-00994-6.

[14] M. Shaikh, A. H. Vayani, S. Akram, and N. Qamar, "Open-source electronic health record systems: A systematic review of most recent advances," Health Informatics Journal, vol. 28, no. 2, 2022, doi: 10.1177/14604582221099828.

[15] R. Casanova, F. A. Villa-Garzon, and J. W. Branch-Bedoya, "Architectural patterns for health information systems: A systematic review," Frontiers in Digital Health, vol. 7, Art. no. 1694839, 2025, doi: 10.3389/fdgth.2025.1694839.