DRS Futures: Technical Principles and Basic Requirements

This document defines the foundational principles for all technical aspects of the DRS Futures project. It encompasses, where relevant, products that we may adopt, or build in house, as well as integration elements between these products.

This document is organized thus:

1. Top-level headers

Identify main technical areas of concern for DRS Futures.

1.1. Second-level headers

Describe the foundational principles qualifying an ideal DRS in such areas.

1.1.1. Third-level headers

Define requirements for achieving those qualities.

- Bullet points under each header represent user stories supporting the requirements and related notes.

Each requirement title has a MoSCoW priority tag between square brackets. Additionally, if the requirement is of Harvard / LTS competency, a [H] is added as well. If the competency is shared between Harvard and the vendor, it is flagged as [H+V]. Otherwise, the vendor is expected to fulfill the requirement.

This document is primarily for the DRS Futures team's use and for sharing with recipients of the RFP, but it may also be shared with stakeholders and other community members for informational purposes.

1. Overall architecture

1.1. Partitioned by concern

1.1.1. [M] Separate preservation from delivery

- [STORY] As a software architect, I want independent and parallel pathways from data sources to delivery and preservation destinations, so that I can independently manage and monitor pathways for multiple purposes.
- [STORY] As a content manager, I want to be able to independently mark a resource for preservation and/or for delivery (ideally to multiple channels), so that I can send the relevant data, and only that, to the right places.

1.1.2. [S] Separate archive from workspace

- [STORY] As a software architect, I want a set of services focused specifically on long-term preservation alongside an independent set of services focused on mid-term, ongoing resource management (workspace), so that: a) I can better design separate workflows and access rules for the two functional areas; and b)I can choose more relevant solutions for each area with less points to compromise on.

- [STORY] As a software architect, I want to offer content versioning support without overwhelming search indices and day-to-day access services with multiple versions of resources, so that I can maintain a fast, efficient staff-facing access and discovery service, while providing access to versioning information when requested.

1.1.3. [M] Independent storage fabric and services

- [STORY] As a repository / DP solution implementer or maintainer, I want to be able to introduce a new service or replace an old one without having to change or migrate the underlying data, so that minimal disruption and cost occurs when upgrading the software architecture.
- [CAVEAT] This limits the software choices available to ones that adhere to a specification, or involves building middleware to adapt the unmatched parts.

1.2. Horizontally scalable

1.2.1. [M] Fault tolerance

- [STORY] As a software architect or service maintainer, I want to ensure that if a node in a complex network of services goes down, more nodes are available to fulfill the same requests of the failed node, so that disruption and down time are minimized.

1.2.2. [M] Statelessness

- [STORY] As an Operations engineer, I want to be able to deploy services in stateless containers, so that I can manage their lifecycle more flexibly without concerns about the data they manage.
- [STORY] As a software engineer, I want to be able to rely on a service API knowing that all the data that service is storing is transactional, while all permanent data are stored in a dedicated storage service or DB, so that: a) I don't have to be concerned about having to share and synchronize data if multiple servers are fulfilling a task; b) I don't have to be concerned about data loss if a server goes down; and c) I have a clear place to look at for data issues.

1.2.3. [M] Plug-in architecture

- [STORY] As a software engineer, I want to be able to replace a piece of software with a different one performing the same tasks, while operating under the same API contract, so that I don't have to maintain irreplaceable legacy software or build "shims" to support a new software solution.
- [STORY] As a software architect, I want to integrate a third-party service that understands the data and protocols used in my architecture, so that I can add services with minimum effort.

1.2.4. [M] Performance responsive to load

- [STORY] As a software architect, I want critical areas of DRS to scale horizontally, e.g., by load- or memory-driven auto-scaling mechanisms, so that I can guarantee a responsive service for a theoretically unlimited amount of input.
- [STORY] As a dev/ops engineer, I want the applications I run to be responsive to enhancements of CPU and memory resources, where appropriate, so that I can easily remove bottlenecks while each service uses resources efficiently.

- [NOTE] There are several limitations to vertical scalability that may or may not be overcome by horizontal scalability (which is considerably more complex).
- [CAVEAT] Achieving higher scalability may have significant developer costs, and should be prioritized only for elements that have been proven to be bottlenecks, possibly via profiling or similar measuring techniques.

1.3. Maintainable

1.3.1. [M] Maximize productivity of code maintenance

- [STORY] As a software developer, I want to work on clearly laid out and automatically tested code, so that I spend as little energy as possible fixing bugs in code that has been deployed to production.
- [STORY] As a software developer, I want to access clear and up to date documentation for the code that I am maintaining, so that I am able to debug it quickly when necessary.
- [NOTE] These user stories are applicable to Harvard developers in the case of open source products.

1.3.2. [M] Stable and healthy code base

- [STORY] As a developer of the core repository system, whether from Harvard, a contracting company or a community contributor, I want to choose libraries and frameworks in a stable status and backed by a wide and active community or internal development team, so that I have more confidence in the reliability of that code.

1.3.3. [M] Configuration-based

- [STORY] As a software developer or architect, I want parts of an application broadly defining the behavior of a service (e.g. defining or modifying content types or processing patterns) be written out in a separate configuration area rather than being hard-wired to the application code, so that changes to the overall behavior can be made with less effort.

1.3.4. [M] Vendor lock-in prevention

1. [STORY] As a software architect, I want to be able to swap products that fulfill a given set of requirements at any time and with minimal or no migration effort, so that I can replace individual products with more cost-effective ones as opportunity arises.
2. [NOTE] This requirement addresses technical concerns. However, there may also be legal or contractual obligations that need to be addressed.

2.1. Service-oriented

1.3.5. [M] API-first

- [STORY] As an API consumer, I want to find an API endpoint offering the same functionality available in a user interface, so that I can more easily automate tasks that users perform manually.
- [STORY] As a QA engineer, I want to ensure that all functionality offered by an application's user interface is built right on top of an API, so that I can predictably and programmatically test the functionality of both with a smaller margin of error.

1.3.6. [S] Adhere to SOA patterns

- [STORY] As a software architect, I want to rely on established techniques and patterns to connect various services, so that I don't have to invent a new and possibly flawed technique that requires a longer learning curve for developers.
- [NOTE] Some SOA patterns may be too rigid for certain scenarios and should be considered as guidelines, not as fixed rules.
- [CAVEAT] Consider https://en.wikipedia.org/wiki/Service-oriented_architecture#Criticism which might conflict with the efficiency principles below.

1.4. Event-driven

1.4.1. [S] Centralize event handling

- [STORY] As a QA engineer or engineering manager, I want to be able to see all the possibly interrelated messages and events at a glance, using one tool, so that I can have a complete view of complex dependencies.
- [STORY] As a QA engineer or engineering manager, I want to be able to subscribe to messages related to any automated process and be notified about failures at any point, so that I don't have to proactively monitor systems or even miss problems for a long time.
- [STORY] As a QA engineer or engineering manager, I want to be able to find a list of failed tasks and retry them with a single click, so that I can remediate issues promptly.

1.4.2. [S] Separate orchestration from task execution

- [STORY] As a software architect, I want to work in an environment where task orchestration is separated from task execution, so that neither application grows uncontrollably as the number and complexity of tasks increases over time.
- [STORY] As a dev/ops engineer, I want to be able to scale up and out the task orchestration service and the workers independently, so that I can manage computing resources more efficiently.

1.4.3. [M] Eventual consistency

- [STORY] As a system administrator, I want to verify that a process, initiated either manually or by event polling and possibly branching into multiple paths and dependencies, has been completed by looking at an administrative dashboard, so that I can easily monitor the health of the system.
- [STORY] As a system administrator, I want to be able to set a timeout for specific tasks so that I would expect that either eventual consistency is achieved within that time limit, or a failure is reported to me, so that I don't need to wait for an indefinite time to check on some operations.
- [STORY] As a data engineer in charge of reprocessing some information resources, I want to be notified about all the stages of a process I expressly initiated, so that I can proceed to further action in a timely manner without having to check the status of my process on my own.
- [NOTE] Shipment carrier notifications which send emails for each step of the fulfillment process are a good example of this approach.

1.4.4. [M] Robust error handling

- [STORY] As a QA engineer or engineering manager, I want a system to automatically retry common temporary failures (e.g. connection errors) for a configurable number of times before they get reported, so that I don't have to be alerted for issues that can fix themselves.

1.5. Continuously improvable

1.5.1. [M] [H] Establish assessment, review and improvement strategy

- [STORY] As a DRS product owner, I want to rely on a clearly defined mechanism for gathering user feedback about the current status of the technology, so that such feedback can be used in periodical assessments, reports, and improvement sprints.
- [STORY] As a DRS product owner, I want to have a clearly defined mechanism to receive periodical assessments from the tech team about areas of improvement of the software architecture, content model, and other internal / technical aspects of DRS, so that I can balance user-facing and technical design improvements accordingly.
- [STORY] As a technical lead, I want to have clear goals to dedicate my development efforts toward, so I can ensure that the maintenance and improvement tasks that my team is fulfilling are going toward the most relevant issues.

2. Access control

2.1. Directed

2.1.1. [M] Internal repository access separated from delivery services

- [STORY] As a software architect, I want to be able to apply separate access policies for internal-only resources and publicly accessible ones.

2.1.2. [M] Internal repository provides metadata for delivery services

- [STORY] As a software architect implementing an access gateway for delivery services, I want to find all necessary access metadata for all publicly accessible resources coming from DRS, so that I can reliably enforce the correct access policies without having to look up and crosswalk resources across several systems.

2.2. Interoperable

2.2.1. [M] Shared knowledge among metadata providers and gateways

- [STORY] As a software or data architect, I want to have one clearly defined, authoritative source for all access-driving metadata, so that I can ensure that access policies are applied based on the most accurate and up-to-date data.

3. Storage

3.1. Purpose-focused

3.1.1. [S] Separate and optimize each storage area for its function

- [STORY] As a software architect, I want to provide the most effective and cost-efficient delivery services for everyday users while maintaining the advantages of a preservation-tuned store, so that the users' day-to-day operations proceed smoothly, and their data are safe.

- [STORY] As a software architect, I want a reasonably fast-access store that is safe for mid-term safekeeping of workspace data and tuned for everyday management operations, along with a store tuned for long-term preservation and sparse access, so that I can better balance resource management for each of the functional areas and reduce costs.
- [STORY] As a software architect, I want the workspace store to be entirely rebuildable from the archival store, so that I can provide a full disaster recovery plan.

3.1.2. [S] Independent pathways to preservation and delivery stores

- [STORY] As a software architect, I want to have clearly independent processes and schedules to synchronize delivery and preservation store targets with their sources, so that there are no operational dependencies for areas that are not functionally interdependent.
- [STORY] As an engineering manager, I want the shortest independent pathways to updating preservation and delivery stores, so that sync operations are faster and the failure of one target does not preclude the success of the other.

3.1.3. [S] Independent layout & structure for preservation and delivery stores

- [STORY] As a software architect, I want to be able to design an access store that is not constrained by the layout chosen for the preservation store, so that I can take advantage of more efficient content storage technologies.
- [STORY] As a business analyst, I want my DRS data to have a replica set in a high-performance data platform, so I have the opportunity to enable BI, ML, and other massive data processing at a later time.

3.2. Technology & platform independent

3.2.1. [S] Communicate over a common protocol

- [STORY] As a repository manager, I want to provide users (including new and external ones unfamiliar with DRS) with a well-known object store API that may abstract any complexity or specific layout of the underlying store, so that adoption is easier.
- [NOTE] This is currently achieved via the S3 protocol, which is implemented in all the individual DRS data stores.

3.2.2. [S] Standards-based

- [STORY] As software architect, I want to ensure that the preservation storage layout is aligned with non-proprietary, openly specified standards, so that:
    1. Our preservation data is decoupled from the software application that manages it at any given time
    2. The engineering team can reuse community tooling to interact with the preservation data in support of use cases outside of the software application that manages the preservation store (System X)

3.2.3. [M] Replication machinery is opaque to the clients

- [STORY] As a repository manager, I want to provide users with only the essential information to be able to do their job, without having to be concerned with how data are replicated behind the scene, so that their interaction with the repository is simpler.

- [STORY] As an Ops manager or software architect, I want to be able to set different replication policies and move replicas across storage tiers/areas without having to coordinate with developers or API clients, so that operations outside my scope of work proceed undisturbed.

3.3. Reliable

3.3.1. [S] Pre-emptive monitoring

- [STORY] As a sysops engineer, I want to be able to run checksum verifications in all the preservation stores, in a routinely or randomized way, using any available checksum of my choice, so that I can perform satisfactory validation of the data at rest that I can balance with resource usage and medium corruption concerns.
- [CAVEAT] Running systematic scans could be expensive, in addition to the fact that “stirring” the preservation layer by continuously running checksums may produce degradation of the medium.

3.3.2. [M] Multiple options for recovery

- [STORY] As a system administrator, I want to be able to rebuild one or many resources that may have become corrupted or deleted from more than one location, so that I can ensure a recovery within multiple realistic disaster scenarios.

4. APIs

4.1. Standards-based

4.1.1. [M] Strive to minimize the number of protocols

- [STORY] As a software architect, I want to ensure that all interaction between general-purpose systems follows the most consistent pattern that is practical to adopt, so that maintenance and documentation is kept to a minimum.
- [NOTE] HTTP/REST is the clear choice here, with obvious exceptions such as AMPQ or STOMP for special-purpose services. Some other protocols might be more efficient in very specific cases, therefore e.g. binary data protocols or the Java API are not excluded a priori; however, these cases should be evaluated carefully and individually.

4.1.2. [M] Strive to adopt existing standards for higher-level functionality

- [STORY] As a software architect, I want to rely on purpose-specific, community-built standards for particular services, so that I don’t have to reinvent the wheel for each one of them.
- [NOTE] Examples may be ActivityStreams for changelogs, Memento for version retrieval, etc.
- [CAVEAT] As with other standards, we should be careful not to shoehorn our requirements into a standard. Sometimes it is best to develop an ad-hoc, locally documented contract among a few systems for efficiency and simplicity purposes than adopting a complex standard that doesn’t completely fit our purpose.

4.2. Meaningfully versioned

4.2.1. [S] Use semantic versioning

- [STORY] As an API client developer, I want to know at a quick glance if an API changed in a way that may be disruptive to software that has been written for a certain API version, so that I can review and update my client code if necessary.

- [STORY] As a release manager, I want to be able to communicate to all maintainers of client applications that use my API that some key methods changed significantly, with a concise and unambiguous message, so that I can better coordinate a new release deployment and minimize disruption.

4.2.2. [M] APIs should include/indicate a specific version

- As a software developer, I want to be informed of the version of the API I am writing a client for, so that I can consult the appropriate documentation and be aware of changes in versions.

4.3. Thoroughly documented

4.3.1. [S] [H] Strive to automate API doc generation

- [STORY] As an engineering manager or QA engineer, I want to ensure that as much of the API documentation as possible is automatically generated, so that discrepancy and obsolescence are minimized.
- [NOTE] Tying doc generation to the CI/CD process is often a good idea.

4.3.2. [M] [H] Build documentation into the development cycles

- [STORY] As an engineering manager, I want to ensure that the quality of API documentation is evaluated during review, so that documentation doesn't become an after-thought.
- [NOTE] Coverage analysis can become part of CI as well; however, too strict QA automation could get in the way more than helping.

5. Content model

5.1. Flexible & extensible

5.1.1. [M] Low barrier to modifying the content model

- [STORY] As a DRS technical resource, I want to be able to add new content types or modify existing ones on end users' request so that content model changes don't require extensive developer effort or long waiting times, so that the semantic quality of the repository can be improved more quickly.
- [NOTE] All content model edits should be mediated through DPS and/or helpdesk in order to maintain overall consistency and good standing.

5.1.2. [S] Prefer network-like relationships over hierarchical structures

- [RATIONALE] Knowledge organization based on parent-child or container-content relationships shows its limitations when capturing complex networks of relationships. A network of connections, which may or may not resemble hierarchical relationships, is more appropriate for the information handled by DRS.
- [STORY] As a metadata librarian, I want to preserve structural metadata about an item that belongs to multiple containers or groups, so that I can reuse that item in multiple contexts.
- [STORY] As a metadata librarian, I want to express the relationship between two items in a way that is less rigid than containment, so I can differentiate users' understanding of the information at hand in a more subtle way.
- [NOTE] A hierarchical approach is still a good fit for modeling content types, which can inherit behavior from each other in a hierarchical fashion.

5.1.3. [M] Clear path for content model migration

- [STORY] As a content manager, I want to re-classify a large number of selected objects to a new content type, so I can take advantage of the better semantics it offers without too much manual work.
- [CAVEAT] This only applies to objects and structures that can be assigned a different content type in a straightforward way, e.g.: if a new specialized image type, geotagged image, is created, and a group of images is assigned that new type; or some opaque content already structured in a way to fit a new content type, that is then assigned in bulk.

5.2. Incremental

5.2.1. [M] Ability to add and modify objects

- [STORY] As a DRS depositor, I want to be able to update an object within the DRS, so that I can keep my collections up to date without having to download the object(s) and re-mint a new URN.

5.3. Versionable

5.3.1. [M] Version creation

- [STORY] As a content manager, I want to be able to create a version of an object when it is modified, so that any changes can be reverted in case of an error, or for historical purposes.

5.3.2. [S] On demand version creation

- [STORY] As a content manager, I want to be able to create a new version of an object only when I decide I want a new one, so that I avoid creating non-meaningful versions every time I save some work in progress.
- [NOTE] This scenario assumes that a 2-layer architecture with a workspace and an archive is adopted, and it would apply to the workspace only. The archive should save a new version each time it receives an update; which means that the archive may be updated less frequently than the workspace.

5.3.3. [M] Version access

- [STORY] As an archivist, I want to retrieve the status of a digital object as of December 12, 2022, regardless of whether it changed or not since, to study its historic context.
- [STORY] As a content manager, I want to inspect the n-th previous version of the metadata of an object, to find the content of a field that at some point got inadvertently deleted.
- [CAVEAT] While version access is very valuable for the stories described above (and more), these cases likely occur quite rarely. Providing versioning in an access store may cause the volume and computational power necessary to run that store to grow exponentially. Unless a strong use case is found for supporting frequent version retrieval, a manual retrieval process for exceptional use cases as described above may be sufficient and would keep resource usage in check.
- [NOTE] This assumes that a separate workspace – archive approach is adopted.

5.3.4. [S] Version retention limiting

- [STORY] As a software architect, I want to be able to set policies for version retention in the archive store, so that I can limit the number of versions retained for each object.

6. Recovery

6.1. Complete

6.1.1. [M] Rebuild operational services

- [STORY] As a support team member charged with disaster recovery tasks, I want to be able to rebuild a whole operational store from the preservation store, without losing any critical data and without having to build any bespoke migration scripts, so that I can guarantee a complete recovery and minimize disruption of users' work.

6.2. Seamlessly integrated in recovery plan

6.2.1. [M] Documented processes for multiple disaster scenarios

- [STORY] As a sysops engineer, I want clear and exhaustive instructions on how to recover specific contents or a whole archive in all foreseeable disaster scenarios, so that I can act swiftly and promptly notify stakeholders of recovery action being taken.
- [NOTE] This documentation is maintained by sysops.

7. Efficiency / effectiveness

7.1. Scalable

7.1.1. [M] Design for future scale

- [STORY] As a DRS stakeholder, I want to plan the deposit of a massive amount of data, several times larger than the current DRS volume, so that all my archives are found in one place.
- [STORY] As a DRS business owner, I want to ensure that a repository expansion by several volume factors is possible without having to redesign the whole architecture, so that I can plan to accommodate the addition of new major collections and a general need for growth with confidence.

7.2. Memory-, I/O-, and computationally-efficient

7.2.1. [M] Tiered storage

- [STORY] As a repository manager, I want to be able to store less frequently used or less critical data in storage segments that use less power and cost less per volume, so I can conserve financial and power resources.
- [STORY] As a repository manager, I want to be able to set rules for data in a way that certain types go automatically into certain storage areas depending on certain properties, so that I can more effectively control the cost of my storage.

7.2.2. [S] [H] Low-footprint applications

- [STORY] As a software engineer or architect, I want to strive to build in-house applications that are inherently efficient, paying particular attention to the algorithms used in the code base, so that I can contain resource usage starting at the bottom of the ecosystem.
- [NOTE] The above implies more developer resources, which present a higher upfront cost because of the need for deeper analysis; not so much in the long term, where well-engineered software usually requires less maintenance on top of being more efficient.

- [CAVEAT] Excessive, preemptive optimization is a notoriously negative approach; optimization efforts should be backed by measurements such as profiling.
- [STORY] As a software adopter, I want to choose products proven to be resource-efficient, so that I can contain resource usage starting at every level of the ecosystem.

7.2.3. [S] [V+H] Scalability should complement efficiency to achieve effectiveness

- [STORY] As a dev/ops engineer or software architect, I want to ensure that the software I am using is already reasonably efficient before planning to allocate more resources to reach a particular metric target, so that I can be resource-conscious rather than resolving bottlenecks by “throwing money” at the problem.

7.3. [H] Measurable performance factors

The following are to be intended as metrics for measuring performance rather than specific targets. A reasonable minimum would be the current DRS metrics, multiplied by a factor that anticipates short- and mid-term growth. The implemented system should offer the possibility of growth beyond that minimum, although not necessarily imlementable in the short term.

7.3.1. Ingest rate

- Objects/time
- Bytes/time

7.3.2. Ingest size

- Single file size, ideally unlimited (i.e. fully streamed)
- Batch size (unlikely unlimited if the batch has to be completely held in memory)

7.3.3. Disaster responsiveness

- Time from corruption to notification
- Time from notification acknowledgement to manual repair (considering size, number of objects, etc.)
- Time from corruption to automatic repair

7.3.4. Cost / volume

- Linear at worst, logarithmic at best
- Exponential is unacceptable

7.3.5. Performance / volume

- Ditto as above

7.3.6. Performance / number of resources

- Ditto as above
- [CAVEAT] Versioning increases the risk of performance degradation becoming exponential with respect to the number of (latest) resources in store. Not indexing historical versions or having a separate, special-access index for them should mitigate this risk.

7.3.7. Response times

- Different expectations depending on the area

7.3.8. [S] [H+V] Environmental footprint assessment

- [STORY] As a DRS product owner, I want access to reliable methodologies to measure the environmental footprint of the service, so that I can better communicate the ethical value of my products if they meet certain standards, and take action if they don't.
- [NOTE] See literature in regard
- [NOTE] There might not be good metrics for this at our disposal. However, a best effort at finding such metrics, even if it concluded that the tools available are inadequate, would constitute valuable information and could spur further development in the field.

8. UI/UX

8.1. Web-based

8.1.1. [S] compatible with major browsers

- [STORY] As a tech support staff member, I want users to move to web-based applications that they can use with standards-compliant web browsers they use daily, so that I don't need to coordinate large-scale deployments of desktop-based applications when DRS undergoes an update.

8.1.2. [S] User interface replicates API functionality

- [STORY] As a software developer who wants to automate some functions available in the DRS UI, I want to have access to the same methods and workflows through an API that a UI user has, so that I can develop my automation tools more accurately and easily.

8.1.3. [S] UI is built on top of the API

- [STORY] As a software engineer or architect or an adopter of a 3rd-party application, I want the application UI to be a thin layer on top of the same functions used by the API, so that the behavior of both is predictable and easier to maintain.

8.1.4. [M] Adhering to established good design practices

- [STORY] As a software engineer or architect, I want to use well-established design practices when designing user interfaces, so that maintenance and team collaboration are simplified.

8.2. Facilitated by automation

8.2.1. [M] Use UI helpers where possible

- [STORY] As a data quality engineer, I want users to be facilitated in executing tasks (e.g. metadata editing) that necessarily require human intervention via UI features, so that less human error caused by repetitive and tedious work occurs.
- [NOTE] Possible UI design improvements: drop-downs from controlled vocabulary items vs. free text fields; batch-fill and batch-duplicate utilities; etc.

9. Process & workflow

9.1. Machine-aided

9.1.1. [M] Automate as many tasks as possible

- [STORY] As a data quality engineer, I want tasks that don’t require human intervention automated so that data issues caused by human error are avoided.

10. Exit strategy

10.1. Comprehensive

10.1.1. [M] Clear path to complete replacement

- [STORY] As a DRS product owner or solutions architect, I want to have a high-level exit strategy plan for DRS as a whole, which considers moving to a completely different storage and software ecosystem, so that I can plan for a possible generational change ahead of time.

10.1.2. [M] Clear path to component replacement

- [STORY] As the DRS product owner, I want a clear path to migrate out of a specific component of DRS, without changing radically the other components of the DRS architecture or migrating contents in the preservation store, so that disruption of the daily operations is minimized, the migration project remains focused on replacing the target component, and the storage fabric is unaffected by this change.
- [STORY] As a DRS software architect, I want to have clear documentation about what replacing each component of the architecture entails, including API endpoints, functional areas, and UI/UX aspects, so that I can more clearly identify the component/s to be replaced and better estimate the effort for a replacement.